

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF: Kazushi HONDA

GAU:

SERIAL NO: 09/605,884

EXAMINER:

FILED: June 29, 2000

FOR: SYSTEM FOR OPTIMIZING DATA TYPE DEFINITION IN PROGRAM LANGUAGE PROCESSING,
METHOD AND COMPUTER READABLE RECORDING MEDIUM THEREFOR

REQUEST FOR PRIORITY

ASSISTANT COMMISSIONER FOR PATENTS
WASHINGTON, D.C. 20231



SIR:

- ☐ Full benefit of the filing date of U.S. Application Serial Number [US App No], filed [US App Dt], is claimed pursuant to the provisions of 35 U.S.C. §120.
- ☐ Full benefit of the filing date of U.S. Provisional Application Serial Number, filed, is claimed pursuant to the provisions of 35 U.S.C. §119(e).
- ☒ Applicants claim any right to priority from any earlier filed applications to which they may be entitled pursuant to the provisions of 35 U.S.C. §119, as noted below.

In the matter of the above-identified application for patent, notice is hereby given that the applicants claim as priority:

COUNTRY

APPLICATION NUMBER

MONTH/DAY/YEAR

JAPAN

11-184252

June 29, 1999

Certified copies of the corresponding Convention Application(s)

- ☒ are submitted herewith
- ☐ will be submitted prior to payment of the Final Fee
- ☐ were filed in prior application Serial No. filed
- ☐ were submitted to the International Bureau in PCT Application Number .
Receipt of the certified copies by the International Bureau in a timely manner under PCT Rule 17.1(a) has been acknowledged as evidenced by the attached PCT/IB/304.
- ☐ (A) Application Serial No.(s) were filed in prior application Serial No. filed ; and
(B) Application Serial No.(s)
 - ☐ are submitted herewith
 - ☐ will be submitted prior to payment of the Final Fee

Respectfully Submitted,

OBLON, SPIVAK, McCLELLAND,
MAIER & NEUSTADT, P.C.

Surinder Sachar

Marvin J. Spivak
Registration No. 24,913

Surinder Sachar
Registration No. 34,423

Fourth Floor
1755 Jefferson Davis Highway
Arlington, Virginia 22202
Tel. (703) 413-3000
Fax. (703) 413-2220
(OSMMN 11/98)

日 本 国 特 許 庁

PATENT OFFICE
JAPANESE GOVERNMENT

別紙添付の書類に記載されている事項は下記の出願書類に記載されて
る事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed
in this Office.

出 願 年 月 日
Date of Application:

1999年 6月29日

願 番 号
Application Number:

平成11年特許願第184252号

願 人
Applicant(s):

株式会社東芝

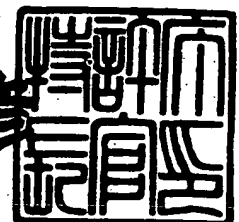


CERTIFIED COPY OF
PRIORITY DOCUMENT

2000年 6月29日

特許庁長官
Commissioner,
Patent Office

近 藤 隆 彦



出証番号 出証特2000-3053750

【書類名】 特許願

【整理番号】 46A98Y069

【提出日】 平成11年 6月29日

【あて先】 特許庁長官殿

【国際特許分類】 G06F 9/00

【発明の名称】 プログラム言語処理システム、そのコード最適化方法、
及び機械読み出し可能な記憶媒体

【請求項の数】 10

【発明者】

 【住所又は居所】 神奈川県川崎市幸区小向東芝町 1 番地 株式会社東芝
 マイクロエレクトロニクスセンター内

 【氏名】 本多 和史

【特許出願人】

 【識別番号】 000003078

 【氏名又は名称】 株式会社 東芝

【代理人】

 【識別番号】 100083806

 【弁理士】

 【氏名又は名称】 三好 秀和

 【電話番号】 03-3504-3075

【選任した代理人】

 【識別番号】 100068342

 【弁理士】

 【氏名又は名称】 三好 保男

【選任した代理人】

 【識別番号】 100100712

 【弁理士】

 【氏名又は名称】 岩▲崎▼ 幸邦

【選任した代理人】

【識別番号】 100100929

【弁理士】

【氏名又は名称】 川又 澄雄

【選任した代理人】

【識別番号】 100108707

【弁理士】

【氏名又は名称】 中村 友之

【選任した代理人】

【識別番号】 100095500

【弁理士】

【氏名又は名称】 伊藤 正和

【選任した代理人】

【識別番号】 100101247

【弁理士】

【氏名又は名称】 高橋 俊一

【選任した代理人】

【識別番号】 100098327

【弁理士】

【氏名又は名称】 高松 俊雄

【手数料の表示】

【予納台帳番号】 001982

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【ブルーフの要否】 要

【書類名】 明細書

【発明の名称】 プログラム言語処理システム、そのコード最適化方法、及び機械読み出し可能な記憶媒体

【特許請求の範囲】

【請求項 1】 原始プログラムの前処理を行う前処理装置と、前処理後の原始プログラムをコンパイルする言語処理装置と、前記原始プログラムを前記前処理装置に翻訳単位で渡し、少なくとも前記前処理装置と前記言語処理装置の間の処理結果の受け渡しを制御するソフトウェア駆動装置とを有するプログラム言語処理システムにおいて、

前記前処理後の原始プログラム全てを走査して、不要なコードを削除し必要な定義のみ実体化するコード最適化処理を行う最適化処理装置を設け、

前記ソフトウェア駆動装置は、

前記前処理後の原始プログラム全てを走査するために、前処理後の原始プログラム全てを前記最適化処理装置に渡す手段と、

前記最適化処理装置による前記コード最適化処理後の各翻訳単位を前記言語処理装置に渡す手段とを有することを特徴とするプログラム言語処理システム。

【請求項 2】 前記最適化処理装置は、

前記前処理後の原始プログラムの中から多相型の定義を検知する多相型定義検知手段と、

前記多相型定義検知手段によって検知された多相型定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第 1 の判定手段と、

前記多相型定義の定義情報が未だ登録されていないと前記第 1 の判定手段によって判定されたときには、前記多相型定義の定義情報を前記多相型定義情報テーブルに登録する第 1 の登録手段と、

前記多相型定義の定義情報が既に登録されていると前記第 1 の判定手段によって判定されたときには、前記多相型定義検知手段によって検知された多相型定義を前記前処理後の原始プログラムの中から削除する第 1 の削除手段と、

前記前処理後の原始プログラムの中から多相型定義の実体生成要求を検知する第 1 の要求検知手段と、

前記第 1 の要求検知手段によって検知された多相型定義の実体生成要求に対応した多相型定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第 2 の判定手段と、

前記多相型定義の実体が未だ生成されていないと前記第 2 の判定手段によって判定されたときには、前記多相型定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第 1 の実体生成手段と、

前記多相型定義の実体が既に生成されていると前記第 2 の判定手段によって判定されたときには、該多相型定義の実体の生成を禁止する第 1 の実体生成禁止手段とを

有することを特徴とする請求項 1 記載のプログラム言語処理システム。

【請求項 3】 前記最適化処理装置は、

前記前処理後の原始プログラムの中から多相型の関数定義を検知する関数定義検知手段と、

前記関数定義検知手段によって検知された多相型関数定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第 3 の判定手段と、

前記多相型関数定義の定義情報が未だ登録されていないと前記第 3 の判定手段によって判定されたときは、前記多相型関数定義の定義情報を前記多相型定義情報テーブルに登録する第 2 の登録手段と、

前記多相型関数定義の定義情報が既に登録されていると前記第 3 の判定手段によって判定されたときには、前記関数定義検知手段によって検知された多相型関数定義を前記前処理後の原始プログラムの中から削除する第 2 の削除手段と、

前記前処理後の原始プログラムの中から多相型関数定義の実体生成要求を検知する第 2 の要求検知手段と、

前記第 2 の要求検知手段によって検知された多相型実体生成要求に対応した多相型関数定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第 4 の判定手段と、

前記多相型関数定義の実体が未だ生成されていないと前記第 4 の判定手段によって判定されたときには、前記多相型関数定義の実体を生成すると共に、この実

体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第 2 の実体生成手段と、

前記多相型の関数定義の実体は既に生成されていると前記第 4 の判定手段によって判定されたときには、前記多相型の関数定義の実体生成を禁止する第 2 の実体生成禁止手段とを

有することを特徴とする請求項 1 または請求項 2 記載のプログラム言語処理システム。

【請求項 4】 前記最適化処理装置は、

前記前処理後の原始プログラムの中から、メンバ関数を保持している多相型を検知するメンバ関数検知手段と、

前記メンバ関数検知手段によって多相型のメンバ関数が検知された場合に、そのメンバ関数を使用されているかどうかを表す使用情報を検知する使用情報手段と、

前記使用情報を多相型定義情報テーブルに登録する第 3 の登録手段と、

前記多相型定義情報テーブルの情報を基に、実際に生成する多相型のメンバ関数定義の実体を決定する決定手段と、

前記決定手段の決定に基づいて、多相型のメンバ関数定義の実体を生成する第 3 の実体生成手段とを

有することを特徴とする請求項 1 乃至請求項 3 記載の言語処理システム。

【請求項 5】 前記最適化処理装置は、

前記前処理後の原始プログラムの中から、データまたは関数の定義を検知する定義検知手段と、

前記定義検知手段で検知されたデータまたは関数の定義に関する定義情報を定義情報テーブルに登録する第 3 の登録手段と、

前記定義情報を基に重複したデータまたは関数の定義を削除する第 3 の削除手段とを

有することを特徴とする請求項 1 乃至請求項 4 記載の言語処理システム。

【請求項 6】 前処理後の原始プログラム全てを走査してコード最適化処理を行う最適化処理装置と、前記最適化処理装置によるコード最適化処理後の各翻

訳単位をコンパイルする言語処理装置と、前記最適化処理装置に対して前処理後の原始プログラム全てを渡すと共に、前記言語処理装置に対して前記コード最適化処理後の各翻訳単位を渡すソフトウェア駆動装置とを有するプログラム言語処理システムのコード最適化方法であって、

前記コード最適化処理は、

前記前処理後の原始プログラムの中から多相型の定義を検知する多相型定義検知処理と、

前記多相型定義検知処理によって検知された多相型定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第 1 の判定処理と、

前記多相型定義の定義情報が未だ登録されていないと前記第 1 の判定処理によって判定されたときには、前記多相型定義の定義情報を前記多相型定義情報テーブルに登録する第 1 の登録処理と、

前記多相型定義の定義情報が既に登録されていると前記第 1 の判定処理によって判定されたときには、前記多相型定義検知処理によって検知された多相型定義を前記前処理後の原始プログラムの中から削除する第 1 の削除処理と、

前記前処理後の原始プログラムの中から多相型定義の実体生成要求を検知する第 1 の要求検知処理と、

前記第 1 の要求検知処理によって検知された多相型定義の実体生成要求に対応した多相型定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第 2 の判定処理と、

前記多相型定義の実体が未だ生成されていないと前記第 2 の判定処理によって判定されたときには、前記多相型定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第 1 の実体生成処理と、

前記多相型定義の実体が既に生成されていると前記第 2 の判定処理によって判定されたときには、前記多相型定義の実体生成を禁止する第 1 の実体生成禁止処理とを

順次実行することを特徴とするプログラム言語処理システムのコード最適化方法。

【請求項 7】 前記コード最適化処理は、

前記前処理後の原始プログラムの中から多相型の関数定義を検知する関数定義検知処理と、

前記関数定義検知処理によって検知された多相型関数定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第 3 の判定処理と、

前記多相型関数定義の定義情報が未だ登録されていないと前記第 3 の判定処理によって判定されたときは、前記多相型関数定義の定義情報を前記多相型定義情報テーブルに登録する第 2 の登録処理と、

前記多相型関数定義の定義情報が既に登録されていると前記第 3 の判定処理によって判定されたときには、前記関数定義検知処理によって検知された多相型関数定義を前記前処理後の原始プログラムの中から削除する第 2 の削除処理と、

前記前処理後の原始プログラムの中から多相型関数定義の実体生成要求を検知する第 2 の要求検知処理と、

前記第 2 の要求検知処理によって検知された多相型実体生成要求に対応した多相型関数定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第 4 の判定処理と、

前記多相型関数定義の実体が未だ生成されていないと前記第 4 の判定処理によって判定されたときには、前記多相型関数定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第 2 の実体生成処理と、

前記多相型の関数定義の実体は既に生成されていると前記第 4 の判定処理によって判定されたときには、前記多相型の関数定義の実体生成を禁止する第 2 の実体生成禁止処理とを

順次実行することを特徴とする請求項 6 記載のプログラム言語処理システムのコード最適化方法。

【請求項 8】 前記コード最適化処理は、

前記前処理後の原始プログラムの中から、メンバ関数を保持している多相型を検知するメンバ関数検知処理と、

前記メンバ関数検知処理によって多相型のメンバ関数が検知された場合に、そ

のメンバ関数が使用されているかどうかを表す使用情報を検知する使用情報処理と、

前記使用情報を多相型定義情報テーブルに登録する第 3 の登録処理と、

前記多相型定義情報テーブルの情報を基に、実際に生成する多相型のメンバ関数定義の実体を決定する決定処理と、

前記決定処理の決定に基づいて、多相型のメンバ関数定義の実体を生成する第 3 の実体生成処理とを

順次実行することを特徴とする請求項 6 または請求項 7 記載の言語処理システムのコード最適化方法。

【請求項 9】 前記コード最適化処理は、

前記前処理後の原始プログラムの中から、データまたは関数の定義を検知する定義検知処理と、

前記定義検知処理で検知されたデータまたは関数の定義に関する定義情報を定義情報テーブルに登録する第 3 の登録処理と、

前記定義情報を基に重複したデータまたは関数の定義を削除する第 3 の削除処理とを

順次実行することを特徴とする請求項 6 乃至請求項 8 記載の言語処理システムのコード最適化方法。

【請求項 10】 請求項 6 乃至請求項 9 記載の言語処理システムのコード最適化方法における前記コード最適化処理を実行するプログラムを格納したことを特徴とする機械読み出し可能な記憶媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、原始プログラムから目的プログラムを生成する等の機能を有するプログラム言語処理システム、そのコード最適化方法、及びこのコード最適化方法を実現するための機械読み出し可能な記憶媒体に関する。

【0002】

【従来の技術】

プログラミング言語で扱うデータは、その採りうる値や操作の適用性などにより様々な型に分類されている。例えば、整数型や実数型、単精度型、倍精度型などの基本データ型のほか、同一または異なる同一、または異なる型で構成される型で構成される構造体を単位とした型も知られている。

【0003】

このように、プログラミング言語の多くは型の概念を導入しており、その中でも、型をパラメータ化することにより、一度表現したアルゴリズムを様々な型に適用できる多相型が注目されている。多相型は、型の族を示すもので、型をパラメータとする型変数を導入することにより、整数型や実数型などという個別の型に依存しない型を定義するものであり、効率的なプログラミングの実現を可能にするものである。

【0004】

図11は、多相型の定義の従来例とその利用方法を示す図であり、図12は、実体化された多相型の定義の従来例を示す図である。

【0005】

図11中に例示する定義101が多相型の定義である。102で示される「T」が型をパラメータとする型変数の宣言であり、103に示すものが実際の型変数である。この多相型の定義101より型変数に適当な型を適用すると、個々の型に応じた実際の定義が生成される。

【0006】

図11に示す104と105が多相型 `vector` のそれぞれ `int` 型（整数型）、`double` 型（倍精度型）に応じた定義の生成を要求しており、図12の106、107に示すような定義の実体が後述の言語処理システムにおいて生成される。

【0007】

図13は、多相型の関数定義の従来例を示す図である。

【0008】

同図の108で示される定義は、多相型の関数を定義している。これは個別の型に依存しない関数を意味する。また、抽象データ型を多相型に適用した場合、

そのデータを扱う関数を特に多相型のメンバ関数と呼ぶ。図 1 3 の 1 0 9 で示される定義は、多相型のメンバ関数を定義している。

【0 0 0 9】

図 1 4 は、従来のプログラム言語処理システムの構成を示すブロック図であり、図 1 5 は、その処理の流れを示す概念図である。

【0 0 1 0】

このプログラム言語処理システムは、原始プログラムから目的プログラムを生成すべく、図 1 4 に示すように、ソフトウェア駆動装置 2 0 1、前処理装置 2 0 3、言語処理装置 2 0 4、及びリンク処理装置 2 0 5 で構成されている。

【0 0 1 1】

ソフトウェア駆動装置 2 0 1 は、原始プログラム群 2 0 2 を取り込み、これを前処理装置 2 0 3、言語処理装置 2 0 4、及びリンク処理装置 2 0 5 に順番に渡すような制御を行い、出力ファイルとして目的プログラムを得ることができる。

即ち、図 1 5 に示すように、各原始プログラム 2 0 2 が対応する前処理装置 2 0 3 にそれぞれ渡されると、前処理が実行されて後段の言語処理装置 2 0 4 に解釈できるような形式のプログラム 2 0 7 がそれぞれ得られる。言語処理装置 2 0 4 は、この各前処理後のプログラム 2 0 7 に対してそれぞれコンパイルを実行し、各リロケータブルオブジェクトファイル 2 0 8 をリンク処理装置 2 0 5 へ出力する。そして、リンク処理装置 2 0 5 が、入力された各リロケータブルオブジェクトファイルをリンクし、これによって目的プログラム 2 0 6 が得られるのである。

【0 0 1 2】

かかるプログラム言語処理システムにおいて多相型が使用されたときは、その多相型の定義の実体を生成し展開することにより多相型のルーチンを実現している。

【0 0 1 3】

例えば、図 1 6 に示す多相型の実現例では、多相型のデータやインラインコードが各翻訳単位毎に必要であるため、多相型が使用された時点 3 1 0 で、機械的に多相型の定義の実体 3 1 1 を翻訳単位に挿入し、多相型名を、指定された型を

基に図中の 312 に示すような適当なルールで生成されたユニークな文字列に変換することで、図 16 では多相型の定義から `int` 型の定義を実現している。

【0014】

【発明が解決しようとする課題】

しかしながら、上記従来のプログラム言語処理システムでは、多相型の関数定義の実体を生成する場合に主に以下の問題点があった。

【0015】

(1) 多相型の関数定義を検知する場合の問題点

多相型の関数の場合では、定義の実体生成が要求される際に、その定義を言語処理システムが検知することは非常に困難である。具体的に説明すると、図 17 に示すような、多相型の宣言ファイル（ヘッダファイル）`stack.h`、多相型の定義ファイル `stack.cpp`、及び多相型を使用するソースファイル `file.cpp` を有する標準的なファイル構成の場合においては、インクルード指令 313, 314 により、ファイル `stack.h` にある多相型の定義の取り込みが行われるが、言語処理システムが多相型の関数定義を検知することができないため、ファイル `stack.cpp` に存在する多相型のメンバ関数定義は、ファイル `file.cpp` には取り込まれないことになる。

【0016】

ここで、宣言 315 により、多相型定義の実体要求が発生する。しかし、ファイル `file.cpp` には翻訳単位中に多相型のメンバ関数定義は存在しないため、言語処理システムは多相型のメンバ関数定義の実体を生成しようと試みるが、定義が別ファイル `stack.cpp` に存在するため、その多相型のメンバ関数定義の実体を生成することができない状況に陥る。

【0017】

したがって、従来では、多相型の関数定義を検索するために、多相型関数定義をヘッダファイルに置くことを要求したり（第 1 の対応策）、または多相型関数定義が置かれるファイル名に制限を設ける（第 2 の対応策）ことなどで対応している。

【0018】

前記第1の対応策では、図18に例示されるように、ヘッダファイル `stack.h` 内に、多相型の宣言だけでなく、316で示された全ての定義を記述されることが要求されている。したがって、318でソースファイル `file.cpp` にある多相型の実体を要求した際に、317で示されるようにそのヘッダファイルがインクルードされていれば、言語処理システムは多相型の関数定義をその翻訳単位で検知することが可能となる。しかしこの場合は、多相型の関数定義を変更する際に必ず再コンパイルを必要とするため、ファイル構成に柔軟さを欠いている。

【0019】

前記第2の対応策では、言語処理システムが多相型の関数定義を検知するために、多相型の関数定義が置かれるファイル名を、言語処理システムが多相型の関数定義を検知するために、ある規則に従った名前（多くは多相型名と同一）にすることを要求する。

【0020】

例えば図19に示すように、ヘッダファイル `stack.h` に置かれた多相型 `Stack` のメンバ関数の定義は、その多相型名 `Stack` と同じソースファイル名 `Stack.cpp` に置くことを原始プログラム作成者に要求する。これによって、宣言320により、319で示される多相型の関数定義「`Stack : Stack`」の実体生成が要求されたときに、決められた名前のファイル `Stack.cpp` 中にその定義を検索し、多相型の関数定義の実体を生成することを可能としている。

【0021】

しかし、この場合は複数の多相型定義をヘッダに記述することができず、多相型の数だけ制限された名前のファイルを作成する必要性が生じ、不必要な手間が原始プログラム作成者に要求される。

【0022】

(2) 不必要な多相型のメンバ関数定義の実体が生成される場合

多相型のメンバ関数の場合、利用されないメンバ関数の実体を生成することはコンパイル時間を増加させ、メモリ領域を余計に使用することになる。具体的に

説明すると、図 2 0 で例示されるような多相型の定義の場合、3 2 1 で示される多相型のメンバ関数の数が 1 0 0 個で、実際に利用される多相型のメンバ関数の数が 1 つだけの場合には、残り 9 9 個の多相型のメンバ関数定義の実体を生成することは明らか効率が悪いことが分かる。

【 0 0 2 3 】

したがって、従来では、効率を無視して全ての多相型のメンバ関数定義の実体を生成するか、利用されている多相型のメンバ関数が全て分かるまで、多相型のメンバ関数定義の実体生成を遅らせることで対応している。

【 0 0 2 4 】

(3) 複数のリロケータブルオブジェクトファイルに多相型の関数定義の実体が生成される場合

多相型の関数定義は、多相型の関数が利用されるたびに実体生成の要求が起きる。したがって、生成要求毎に多相型の関数定義の実体を生成すると、複数の同じ定義が存在することになる。例えば図 2 1 で示されるように、同一翻訳単位内 3 2 2, 3 2 3 で多相型の関数定義の実体を生成する場合には、言語処理システムがその多相型の関数定義が実体化されたかどうかのフラグ変数を持つことで、重複した定義の生成を回避することが可能である。

【 0 0 2 5 】

しかし、別の翻訳単位で 3 2 4 に示されるような多相型の関数定義の実体生成要求がある場合には、従来の言語処理システムが他の翻訳単位で既にその多相型の関数定義が生成されているかどうかを判別することができないという問題がある。

【 0 0 2 6 】

したがって、従来では、原始プログラム作成者が特別な前処理指令を用いて重複した多相型の関数定義の実体生成を抑制する方法や、リンク処理時にリンクが重複した多相型の関数定義を削除する方法で対応しているが、前者は原始プログラム作成者に不要な制限を要求しており、後者はその判別処理に多大な時間を必要としている。

【 0 0 2 7 】

本発明は、上述の如き従来の問題点を解決するためになされたもので、その目的は、多相型の関数定義を容易に検知することができ、しかも重複した不要な多相型の定義を取り除くことができると共に、不要な多相型の実体化を回避することができるプログラム言語処理システム、そのコード最適化方法、及びこのコード最適化方法を実現するための機械読み出し可能な記憶媒体を提供することである。

【0028】

【課題を解決するための手段】

上記目的を達成するために、請求項1記載の発明に係るプログラム言語処理システムでは、原始プログラムの前処理を行う前処理装置と、前処理後の原始プログラムをコンパイルする言語処理装置と、前記原始プログラムを前記前処理装置に翻訳単位で渡し、少なくとも前記前処理装置と前記言語処理装置の間の処理結果の受け渡しを制御するソフトウェア駆動装置とを有するプログラム言語処理システムにおいて、前記前処理後の原始プログラム全てを走査して、不要なコードを削除し必要な定義のみ実体化するコード最適化処理を行う最適化処理装置を設け、前記ソフトウェア駆動装置は、前記前処理後の原始プログラム全てを走査するために、前処理後の原始プログラム全てを前記最適化処理装置に渡す手段と、前記最適化処理装置による前記コード最適化処理後の各翻訳単位を前記言語処理装置に渡す手段とを有することを特徴とする。

【0029】

請求項2記載の発明に係るプログラム言語処理システムでは、請求項1記載のプログラム言語処理システムにおいて、前記最適化処理装置は、前記前処理後の原始プログラムの中から多相型の定義を検知する多相型定義検知手段と、前記多相型定義検知手段によって検知された多相型定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第1の判定手段と、前記多相型定義の定義情報が未だ登録されていないと前記第1の判定手段によって判定されたときには、前記多相型定義の定義情報を前記多相型定義情報テーブルに登録する第1の登録手段と、前記多相型定義の定義情報が既に登録されていると前記第1の判定手段によって判定されたときには、前記多相型定義検知手段によって検知

された多相型定義を前記前処理後の原始プログラムの中から削除する第 1 の削除手段と、前記前処理後の原始プログラムの中から多相型定義の実体生成要求を検知する第 1 の要求検知手段と、前記第 1 の要求検知手段によって検知された多相型定義の実体生成要求に対応した多相型定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第 2 の判定手段と、前記多相型定義の実体が未だ生成されていないと前記第 2 の判定手段によって判定されたときには、前記多相型定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第 1 の実体生成手段と、前記多相型定義の実体が既に生成されていると前記第 2 の判定手段によって判定されたときには、該多相型定義の実体生成を禁止する第 1 の実体生成禁止手段とを有することを特徴とする。

【 0 0 3 0 】

請求項 3 記載の発明に係るプログラム言語処理システムでは、請求項 1 または請求項 2 記載のプログラム言語処理システムにおいて、前記最適化処理装置は、前記前処理後の原始プログラムの中から多相型の関数定義を検知する関数定義検知手段と、前記関数定義検知手段によって検知された多相型関数定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第 3 の判定手段と、前記多相型関数定義の定義情報が未だ登録されていないと前記第 3 の判定手段によって判定されたときは、前記多相型関数定義の定義情報を前記多相型定義情報テーブルに登録する第 2 の登録手段と、前記多相型関数定義の定義情報が既に登録されていると前記第 3 の判定手段によって判定されたときには、前記関数定義検知手段によって検知された多相型関数定義を前記前処理後の原始プログラムの中から削除する第 2 の削除手段と、前記前処理後の原始プログラムの中から多相型関数定義の実体生成要求を検知する第 2 の要求検知手段と、前記第 2 の要求検知手段によって検知された多相型実体生成要求に対応した多相型関数定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第 4 の判定手段と、前記多相型関数定義の実体が未だ生成されていないと前記第 4 の判定手段によって判定されたときには、前記多相型関数定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として

前記多相型定義情報テーブルに登録する第 2 の実体生成手段と、前記多相型の関数定義の実体は既に生成されていると前記第 4 の判定手段によって判定されたときには、前記多相型の関数定義の実体生成を禁止する第 2 の実体生成禁止手段とを有することを特徴とする。

【0031】

請求項 4 記載の発明に係るプログラム言語処理システムでは、請求項 1 乃至請求項 3 記載の言語処理システムにおいて、前記最適化処理装置は、前記前処理後の原始プログラムの中から、メンバ関数を保持している多相型を検知するメンバ関数検知手段と、前記メンバ関数検知手段によって多相型のメンバ関数が検知された場合に、そのメンバ関数が使用されているかどうかを表す使用情報を検知する使用情報手段と、前記使用情報を多相型定義情報テーブルに登録する第 3 の登録手段と、前記多相型定義情報テーブルの情報を基に、実際に生成する多相型のメンバ関数定義の実体を決定する決定手段と、前記決定手段の決定に基づいて、多相型のメンバ関数定義の実体を生成する第 3 の実体生成手段とを有することを特徴とする。

【0032】

請求項 5 記載の発明に係るプログラム言語処理システムでは、請求項 1 乃至請求項 4 記載の言語処理システムにおいて、前記最適化処理装置は、前記前処理後の原始プログラムの中から、データまたは関数の定義を検知する定義検知手段と、前記定義検知手段で検知されたデータまたは関数の定義に関する定義情報を定義情報テーブルに登録する第 3 の登録手段と、前記定義情報を基に重複したデータまたは関数の定義を削除する第 3 の削除手段とを有することを特徴とする。

【0033】

請求項 6 記載の発明に係るプログラム言語処理システムのコード最適化方法では、前処理後の原始プログラム全てを走査してコード最適化処理を行う最適化処理装置と、前記最適化処理装置によるコード最適化処理後の各翻訳単位をコンパイルする言語処理装置と、前記最適化処理装置に対して前処理後の原始プログラム全てを渡すと共に、前記言語処理装置に対して前記コード最適化処理後の各翻訳単位を渡すソフトウェア駆動装置とを有するプログラム言語処理システムのコ

ード最適化方法であって、前記コード最適化処理は、前記前処理後の原始プログラムの中から多相型の定義を検知する多相型定義検知処理と、前記多相型定義検知処理によって検知された多相型定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第 1 の判定処理と、前記多相型定義の定義情報が未だ登録されていないと前記第 1 の判定処理によって判定されたときには、前記多相型定義の定義情報を前記多相型定義情報テーブルに登録する第 1 の登録処理と、前記多相型定義の定義情報が既に登録されていると前記第 1 の判定処理によって判定されたときには、前記多相型定義検知処理によって検知された多相型定義を前記前処理後の原始プログラムの中から削除する第 1 の削除処理と、前記前処理後の原始プログラムの中から多相型定義の実体生成要求を検知する第 1 の要求検知処理と、前記第 1 の要求検知処理によって検知された多相型定義の実体生成要求に対応した多相型定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第 2 の判定処理と、前記多相型定義の実体が未だ生成されていないと前記第 2 の判定処理によって判定されたときには、前記多相型定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第 1 の実体生成処理と、前記多相型定義の実体が既に生成されていると前記第 2 の判定処理によって判定されたときには、前記多相型定義の実体生成を禁止する第 1 の実体生成禁止処理とを順次実行することを特徴とする。

【 0 0 3 4 】

請求項 7 記載の発明に係るプログラム言語処理システムのコード化最適化方法では、請求項 6 記載のプログラム言語処理システムのコード化最適化方法において、前記コード最適化処理は、前記前処理後の原始プログラムの中から多相型の関数定義を検知する関数定義検知処理と、前記関数定義検知処理によって検知された多相型関数定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第 3 の判定処理と、前記多相型関数定義の定義情報が未だ登録されていないと前記第 3 の判定処理によって判定されたときは、前記多相型関数定義の定義情報を前記多相型定義情報テーブルに登録する第 2 の登録処理と、前記多相型関数定義の定義情報が既に登録されていると前記第 3 の判定処理によっ

て判定されたときには、前記関数定義検知処理によって検知された多相型関数定義を前記前処理後の原始プログラムの中から削除する第 2 の削除処理と、前記前処理後の原始プログラムの中から多相型関数定義の実体生成要求を検知する第 2 の要求検知処理と、前記第 2 の要求検知処理によって検知された多相型実体生成要求に対応した多相型関数定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第 4 の判定処理と、前記多相型関数定義の実体が未だ生成されていないと前記第 4 の判定処理によって判定されたときには、前記多相型関数定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第 2 の実体生成処理と、前記多相型の関数定義の実体は既に生成されていると前記第 4 の判定処理によって判定されたときには、前記多相型の関数定義の実体生成を禁止する第 2 の実体生成禁止処理とを順次実行することを特徴とする。

【 0 0 3 5 】

請求項 8 記載の発明に係るプログラム言語処理システムのコード化最適化方法では、請求項 6 または請求項 7 記載の言語処理システムのコード最適化方法において、前記コード最適化処理は、前記前処理後の原始プログラムの中から、メンバ関数を保持している多相型を検知するメンバ関数検知処理と、前記メンバ関数検知処理によって多相型のメンバ関数が検知された場合に、そのメンバ関数を使用されているかどうかを表す使用情報を検知する使用情報処理と、前記使用情報を多相型定義情報テーブルに登録する第 3 の登録処理と、前記多相型定義情報テーブルの情報を基に、実際に生成する多相型のメンバ関数定義の実体を決定する決定処理と、前記決定処理の決定に基づいて、多相型のメンバ関数定義の実体を生成する第 3 の実体生成処理とを順次実行することを特徴とする。

【 0 0 3 6 】

請求項 9 記載の発明に係るプログラム言語処理システムのコード化最適化方法では、請求項 6 乃至請求項 8 記載の言語処理システムのコード最適化方法において、前記コード最適化処理は、前記前処理後の原始プログラムの中から、データまたは関数の定義を検知する定義検知処理と、前記定義検知処理で検知されたデータまたは関数の定義に関する定義情報を定義情報テーブルに登録する第 3 の登

録処理と、前記定義情報を基に重複したデータまたは関数の定義を削除する第 3 の削除処理とを順次実行することを特徴とする。

【0037】

請求項 10 記載の発明に係る機械読み出し可能な記憶媒体では、請求項 6 乃至請求項 9 記載の言語処理システムのコード最適化方法における前記コード最適化処理を実行するプログラムを格納したことを特徴とする。

【0038】

【発明の実施の形態】

以下、本発明の実施の形態を図面に基づいて説明する。

【0039】

[第 1 実施形態]

図 1 は、本発明の第 1 実施形態に係るプログラム言語処理システムの基本構成を示すブロック図である。

【0040】

このプログラム言語処理システムは、原始プログラム群 2 から目的プログラム 7 を生成すべく、ソフトウェア駆動装置 1、前処理装置 3、最適化処理装置 4、言語処理装置 5、及びリンク処理装置 6 で構成されている。そのうち、前処理装置 3、言語処理装置 5 及びリンク処理装置 6 は、図 14 に示した従来システムにおける前処理装置 203、言語処理装置 204 及びリンク処理装置 205 と同一の機能を有するものである。従って、本実施形態のプログラム言語処理システムの構成では、図 14 の従来構成の言語処理システムにおいて、新たに最適化処理装置 4 を設け、さらにこれに対応して、ソフトウェア駆動装置 201 の機能を拡張したソフトウェア駆動装置 1 を設けた点が特徴となっている。

【0041】

即ち、本実施形態のプログラム言語処理システムは、例えばコンパイラなど、CPU のオブジェクトコードを生成するシステムであり、多相型を扱うことができ、前処理が終了した全ての入力プログラムを最適化処理装置 4 に渡すことができるように制御可能なソフトウェア駆動装置 1 を有している。最適化処理装置 4 は、全ての入力プログラムに対し多相型定義情報テーブル 4a を作成し、そのテ

ーブル 4 a に格納された情報により不要なコードを削除することで、コードの最適化を実施する。

【0042】

以下、本実施形態の特徴部分について詳細に説明する。

【0043】

図 2 は、本実施形態に係るプログラム言語処理システムの処理の流れを示す概念図である。

【0044】

ソフトウェア駆動装置 1 は、取り込んだ原始プログラム群 2 を、前処理装置 3、最適化処理装置 4、言語処理装置 5、及びリンク処理装置 6 に順番に渡すような制御を行う。

【0045】

原始プログラム群 2 がソフトウェア駆動装置 1 により各ファイル毎に前処理装置 3 に渡されると、前処理装置 3 による前処理指令の実行により従来の言語処理システムと同様の前処理が行われる。その結果、前処理装置 3 からは、前処理後のファイル 8 が各翻訳単位として出力される。ここで、全ての前処理後のファイル 8 が最適化処理装置 4 に入力され、後述するコード最適化処理が全ての入力ファイルに対して完了した後に、各々の翻訳単位が平行に言語処理装置 5 に振り分けられる。ここで、最適化処理装置 4 への前処理後の全ファイルの読み込み制御と、言語処理装置 5 への最適化処理後のファイルの振り分け制御は、ソフトウェア駆動装置 1 によって行われ、このソフトウェア駆動装置 1 の機能が従来システムのソフトウェア駆動装置 201 に対して拡張された機能となっている。

【0046】

以降の処理は、従来の言語処理システムと同様に各言語処理装置 5 が生成したリロケータブルオブジェクト 9 をリンク処理装置 6 でリンクして、目的プログラム 7 が得られる。

【0047】

次に、本実施形態のコード最適化処理を図 3、図 4 及び図 5 を参照して説明する。なお、図 3 は、本実施形態に係るコード最適化処理を示すフローチャートで

ある。また、図 4 は、本実施形態に係る、多相型定義情報テーブル 4 a への多相型定義の登録例を示す図であり、図 5 は、本実施形態に係る多相型定義の実体生成例を示す図である。

【0048】

図 3 において、まず、前述したように、ソフトウェア駆動装置 1 が原始プログラム 2 を前処理装置 3 に渡し（ステップ S 1 1）、前処理装置 3 が前処理指令を実行して前処理が行われると（ステップ S 1 2）、ソフトウェア駆動装置 1 は、前処理後のプログラム f 0 を最適化処理装置 4 に渡す（ステップ S 1 3）。

【0049】

最適化処理装置 4 は、コードの最適化を実行するために、今回入力された前処理後のプログラム f 0 を走査し、当該入力プログラム f 0 中の多相型の定義を検索する（ステップ S 1 4）。このとき、多相型定義が存在しないときは、後述するステップ S 1 8 の多相型実体要求の検索処理に移行する。

【0050】

多相型の定義が存在すれば、その定義が既にあるかどうかを多相型定義情報テーブル 4 a の定義情報を基に調べ（ステップ S 1 5）、未だ定義されていない多相型定義であれば、多相型定義情報テーブル 4 a にその定義情報を登録する（ステップ S 1 6）。

【0051】

この多相型定義情報テーブル 4 a への登録処理では、検知した多相型定義の型名「Tree」38 を定義情報として、多相型定義情報テーブル 4 a に登録する（図 4 の 4 0 参照）。そして、このとき、多相型「Tree」の定義が今回のプログラム f 0 上で使用されていれば（図 4 中の 3 9 参照）、多相型定義情報テーブル 4 a 上の多相型定義情報の使用フラグを使用状態に設定する（図 4 の 4 1 参照）。

【0052】

一方、既に定義されている多相型の定義であれば、その定義は重複していることになるため、多相型定義情報テーブル 4 a にはその定義情報を登録せず、重複した当該多相型の定義を入力プログラム f 0 中から削除する（ステップ S 1 7）

。即ち、既に、型名「Tree」が多相型定義情報テーブル4aに登録されていれば、今回検知された多相型定義37は重複した定義とみなし、今回読み込まれた入力プログラムf0上から削除するのである。

【0053】

また、図3で示した本実施形態のコード最適化処理の実行時に、多相型定義情報テーブル4aに登録された型名で、その型名が全プログラム中で使用されていないと判明したときは（前記使用フラグが未使用状態）、最適化処理装置4は、当該多相型定義を不要とみなし、入力プログラム上から該多相型定義を削除する。かようにして多相型定義の走査が完了した後、最適化処理装置4は、多相型の使用による多相型定義の実体化要求を検索する（ステップS18）。このとき、多相型定義の実体化要求が存在すれば、その実体化された多相型定義の型が既に存在するかどうかを、多相型定義情報テーブル4aを基に調べ（ステップS19）、未だ実体化されていない多相型定義であれば、多相型の実体（定義）を生成する（ステップS20）。一方、既に実体化された多相型定義であれば、その実体化された定義は重複することになるため、多相型の実体は生成しない（ステップS21）。

【0054】

この多相型実体生成処理について図5を用いて具体的に説明する。最適化処理装置4が、例えばint型を使用する多相型実体生成要求45を検知すると、その多相型定義の実体が既に生成されているかどうかを多相型定義情報テーブル4aに問い合わせる。このとき、未だ実体が生成されていなければ、43で示すint型の多相型定義の実体を生成し、多相型定義情報テーブル4aには既に実体を生成した旨の情報が登録される。この際、多相型の名前は、プログラム中でユニークな名前に変換され、int型を使用する多相型実体生成要求45などの全ての型名も、48で示すようにユニークな名前に変換される。

【0055】

同様に、double型を使用する多相型実体生成要求46が検知されると、44で示すdouble型の多相型定義の実体生成、多相型定義情報テーブル4aへの情報登録、及び49、50で示すようなユニークな名前への変換が行われ

る。ここで、47で示すような、double型を使用する多相型定義の実体生成要求を検知した場合は、多相型定義情報テーブル4aには「既にdouble型の多相型定義の実体は生成済み」との情報があるため、この情報に基づき44で示すようなdouble型を使用する多相型定義の実体は生成しない。

【0056】

また、多相型定義の実体を生成する際に、もしint型とdouble型が同じ型のアーキテクチャであるような場合には、多相型定義の実体は同じものとみなされ、多相型定義の実体は1つしか生成されない。以上の結果、図5に示すような最適化後の原始プログラムf0'が得られる。

【0057】

これ以降は、上記の処理を全ての原始プログラムに対して繰り返し（ステップS22）、処理が終了した各プログラムを、ソフトウェア駆動装置1により言語処理装置5に渡すことになる（ステップS23）。

【0058】

このように本実施形態では、最適化処理装置4が、上記コード最適化処理を全ての原始プログラムに対して実行するので、全ての原始プログラム内の多相型の定義情報と多相型の実体化要求情報とを収集することができる。これにより、重複した不要な多相型定義を取り除くことができると共に、不要な多相型の実体生成を回避することができるため、オブジェクトサイズを小さくし、実行速度を向上させることが可能になる。また、コンパイル以前に最適化を完了するため、多相型にかかるコンパイル・リンク時間の高速化が図られる。

【0059】

[第2実施形態]

第2実施形態では、図1、図2、及び図3で説明した上記第1実施形態の構成において、多相型の関数定義に関して、特に多相型定義情報テーブルへの登録、及び実体生成について説明するものである。

【0060】

図6は、本実施形態に係る多相型関数定義の多相型定義情報テーブル4aへの登録例を説明するための図である。

【0061】

最適化処理装置4は、前処理後の原始プログラムf1とf2を読み込み（図3のステップS13）、多相型の関数定義51を検知すると（図3のステップS14）、その関数定義51が既にあるかどうかを多相型定義情報テーブル4aの定義情報を基に調べる（図3のステップS15）。

【0062】

未だ定義されていない多相型の関数定義であれば、検知した多相型の型名「max」52を多相型定義情報テーブル4aに登録する。もしこのとき、既に型名「max」52が多相型定義情報テーブル4aに登録されていれば、この多相型の関数定義51は重複した定義とみなし、前処理後の原始プログラムf1上から削除する（図3のステップS17）。

【0063】

さらに、図6の53、54、55のように多相型51が使用されていれば、それぞれ多相型定義の実体要求が生じた型毎に、56、57で示される多相型定義情報テーブル4a上の多相型定義情報の使用フラグを使用状態に設定する。

【0064】

また、多相型定義情報テーブル4aに登録された多相型の関数名で、その関数名が全プログラム中で使用されていなければ、最適化処理装置4はその多相型の定義を不要とみなし、前処理後の原始プログラム上から定義を削除する。

【0065】

図7は、本実施形態に係る多相型関数定義の実体生成例を示す図である。

【0066】

最適化処理装置4は、図6で説明したように多相型の関数定義51を読み込み、多相型定義情報テーブル4aに登録する。その後、int型を使用する多相型の関数定義の実体生成要求53を検知すると（図3のステップS18）、多相型定義情報テーブル4aにその多相型の関数定義の実体が既に生成されているかどうか問い合わせる（図3のステップS19）。

【0067】

このとき未だ実体が生成されていなければ、62で示すint型を使用する多

相型の関数定義の実体を生成し（図 3 のステップ S 2 0）、多相型定義情報テーブル 4 a には既に実体を生成した旨の情報が登録される。この際、多相型関数の名前はプログラム中でユニークな名前に変換され、その関数を使用する 6 4, 6 6 などに示す全ての関数名がそのユニークな名前に変換される。

【 0 0 6 8 】

同様に、double 型を使用する多相型の関数定義の実体生成要求 5 4 が検知されると、6 3 で示す double 型を使用する多相型の関数定義の実体を生成し（図 3 のステップ S 2 0 に相当）、多相型定義情報テーブル 4 a への実体生成済みとの情報登録と、6 5 で示すようなユニークな名前への変換とが行われる。

【 0 0 6 9 】

ここで、6 1 で示す int 型を使用する多相型の関数定義の実体生成要求を検知した場合、多相型定義情報テーブル 4 a には、int 型を使用する多相型の関数定義の実体は既に生成済みとの情報があるため、6 2 で示すような int 型を使用する多相型の関数定義の実体を生成することは行わない（図 3 のステップ S 2 1）。

【 0 0 7 0 】

また、多相型関数の定義の実体を生成する際に、もし int 型と double 型が同じ型のアーキテクチャであるような場合には、多相型関数の定義の実体は同じものとみなされ、多相型関数の定義の実体は 1 つしか生成されない。

【 0 0 7 1 】

以上の結果、図 7 に示すような最適化後の原始プログラム f 1' , f 2' が得られる。

【 0 0 7 2 】

このように、本実施形態においても、多相型の関数定義において、上記第 1 実施形態と同等の効果を得ることができる。

【 0 0 7 3 】

〔第 3 実施形態〕

第 3 実施形態では、図 1、図 2、及び図 3 で説明した上記第 1 実施形態の構成

において、多相型のメンバ関数定義に関して、特に多相型定義情報テーブルへの登録、及び実体生成について説明するものである。

【 0 0 7 4 】

図 8 は、本実施形態に係る多相型メンバ関数定義の多相型定義情報テーブル 4 a への登録例を示す図である。また、図 9 は、本実施形態に係る多相型メンバ関数の最適化処理を示すフローチャートであり、この処理は、図 3 におけるステップ S 2 0 における多相型の実体（定義）生成処理時に実施される。

【 0 0 7 5 】

最適化処理装置 4 は、図 8 に示された前処理後の原始プログラム f 3 を読み込み（図 3 のステップ S 1 3）、多相型の定義 6 6 を検知する（図 3 のステップ S 1 4）。このとき、多相型がメンバ関数を持つならば、最適化処理装置 4 は、6 7 に示す多相型のメンバ関数が 6 8、6 9 のように使用されているかの情報を同時に検知し（図 3 のステップ S 1 5）、多相型のメンバ関数使用情報を 7 0、7 1 のように多相型定義情報テーブル 4 a に登録し（図 3 のステップ S 1 6）、7 2 のように使用フラグを使用状態に設定する。

【 0 0 7 6 】

その後、最適化処理装置 4 が、多相型の実体生成要求 7 6 を検知すると（図 3 のステップ S 1 8）、多相型定義情報テーブル 4 a に問い合わせ、未だ当該多相型定義の実体が生成されていなければ（図 3 のステップ S 1 9）、この多相型定義の実体（定義）生成を行う（ステップ S 2 0）。

【 0 0 7 7 】

この多相型の実体（定義）の生成時に、最適化処理装置 4 は、図 9 のフローチャートに示す処理を実施する。図 9 において、まず全ての多相型のメンバ関数で必要となるメンバ変数定義の実体化を行う（ステップ S 3 1）。

【 0 0 7 8 】

次いで、多相型がメンバ関数を持つならば、そのメンバ関数を使用されているかどうかを多相型定義情報テーブル 4 a に問い合わせ（ステップ S 3 2、ステップ S 3 3）、使用されている場合はその多相型メンバ関数定義の実体を生成する（ステップ S 3 4）。また使用されていない場合は、その多相型メンバ関数の定

義の実体は生成しない。

【0 0 7 9】

この処理を全ての多相型のメンバ関数について調べる（ステップ S 3 5）。これにより、本実施形態では、使用されていない多相型のメンバ関数定義の実体を生成するのを未然に回避することができる。

【0 0 8 0】

図 1 0 は、本実施形態に係る多相型のメンバ関数定義の実体生成例を示す図である。

【0 0 8 1】

最適化処理装置 4 が、例えば i n t 型の多相型実体生成要求 7 6 を検知すると、多相型定義情報テーブル 4 a に問い合わせ、未だ当該多相型定義の実体が生成されていなければ、多相型のメンバ関数の使用情報を基に、実際に生成する多相型のメンバ関数を決定する。

【0 0 8 2】

図 1 0 に示した例では、i n t 型の実体で使用される多相型のメンバ関数定義 7 4 は 2 つであるため、実際に i n t 型の多相型定義 7 8 で実体が生成されるメンバ関数は 7 9 に示すようになり、他の使用されていない多相型のメンバ関数定義の実体は生成されない。

【0 0 8 3】

同様に d o u b l e 型の多相型の実体生成要求 7 7 が検知されると、未だその多相型定義の実体が生成されていなければ、多相型定義情報テーブル 4 a より多相型のメンバ関数の使用情報を基に実際に生成する多相型のメンバ関数 7 5 が 1 つだけであると決定する。この決定に従い、実際に d o u b l e 型の多相型の定義 8 0 で実体が生成されるメンバ関数は 8 1 で示す 1 つだけとなる。

【0 0 8 4】

以上の結果、図 1 0 に示すような最適化後の原始プログラム f 3' が得られる。

【0 0 8 5】

このように、本実施形態においても、多相型のメンバ関数定義において、上記

第1実施形態と同等の効果を得ることができる。

【0086】

なお、上記第1、第2、及び第3実施形態では、多相型で説明をしたが、図1及び図2に示した構成を利用することにより、多相型に限らず、原始プログラム群中における重複したデータ、重複した関数を削除することができる。

【0087】

この場合も、多相型同様に最適化処理装置4が前処理後の原始プログラムを読み込み、各データ、関数の定義を定義情報テーブル4aに登録する。このとき、定義情報テーブル4aの情報を基に既に定義されているデータや関数を最適化処理装置4が検知した場合、このデータや関数の定義は重複した定義とみなし、前処理後のプログラムから削除する。また、定義情報テーブル4aに登録されたデータや関数の定義で、そのデータや関数が全プログラム中で使用されていなければ、最適化処理装置4はその定義を不要とみなし、前処理後の原始プログラム上から定義を削除する。

【0088】

また、上記第1、第2、及び第3実施形態の各機能を実現するためのプログラムをCD-ROMなどの記憶媒体に格納することで、本発明を実現することも可能である。

【0089】

【発明の効果】

以上詳細に説明したように、請求項1記載の発明によれば、不要なコードを削除し必要な定義のみ実体化することができるので、コードサイズを縮小することができ、実行速度が向上する。さらに、コンパイル以前にコード最適化処理が完了するため、コンパイルやリンク時間の高速化が実現する。

【0090】

請求項2記載の発明によれば、請求項1の発明において、重複した不要な多相型の定義を取り除くことができると共に、不要な多相型の実体化を回避することができ、多相型に関して上記請求項1の発明と同等の効果を奏する。

【0091】

請求項 3 記載の発明によれば、請求項 1 または請求項 2 の発明において、多相型の関数定義を容易に検知することができ、しかも重複した不要な多相型関数の定義を取り除くことができると共に、不要な多相型関数の実体化を回避することができ、多相型に関して上記請求項 1 の発明と同等の効果を奏する。

【0092】

請求項 4 記載の発明によれば、請求項 1 乃至請求項 3 の発明において、重複した不要な多相型のメンバ関数定義を取り除くことができると共に、不要な多相型メンバ関数の実体化を回避することができ、多相型に関して上記請求項 1 の発明と同等の効果を奏する。

【0093】

請求項 5 記載の発明によれば、請求項 1 乃至請求項 4 の発明において、データまたは関数の定義の重複を防ぐことができ、全てのデータ型に関して上記請求項 1 の発明と同等の効果を奏する。

【0094】

請求項 6 記載の発明によれば、重複した不要な多相型の定義を取り除くことができると共に、不要な多相型の実体化を回避することができ、多相型に関して上記請求項 1 の発明と同等の効果を奏する。

【0095】

請求項 7 記載の発明によれば、請求項 6 の発明において、多相型の関数定義を容易に検知することができ、しかも重複した不要な多相型関数の定義を取り除くことができると共に、不要な多相型関数の実体化を回避することができ、多相型に関して上記請求項 1 の発明と同等の効果を奏する。

【0096】

請求項 8 記載の発明によれば、請求項 6 または請求項 7 の発明において、重複した不要な多相型のメンバ関数定義を取り除くことができると共に、不要な多相型メンバ関数の実体化を回避することができ、多相型に関して上記請求項 1 の発明と同等の効果を奏する。

【0097】

請求項 9 記載の発明によれば、請求項 6 乃至請求項 8 の発明において、データ

または関数の定義の重複を防ぐことができ、全てのデータ型に関して上記請求項 1 の発明と同等の効果を奏する。

【0 0 9 8】

請求項 1 0 記載の発明によれば、請求項 6 乃至請求項 9 の発明と同等の効果を奏する。

【図面の簡単な説明】

【図 1】

本発明の第 1 実施形態に係るプログラム言語処理システムの基本構成を示すブロック図である。

【図 2】

第 1 実施形態に係るプログラム言語処理システムの処理の流れを示す概念図である。

【図 3】

第 1 実施形態に係る最適化処理を示すフローチャートである。

【図 4】

第 1 実施形態に係る、多相型定義情報テーブルへの多相型定義の登録例を示す図である。

【図 5】

第 1 実施形態に係る多相型定義の実体生成例を示す図である。

【図 6】

第 2 実施形態に係る多相型関数定義の多相型定義情報テーブルへの登録例を説明するための図である。

【図 7】

第 2 実施形態に係る多相型関数定義の実体生成例を示す図である。

【図 8】

第 3 実施形態に係る多相型メンバ関数定義の多相型定義情報テーブル 4 a への登録例を示す図である。

【図 9】

第 3 実施形態に係る多相型メンバ関数の最適化処理を示すフローチャートであ

る。

【図 1 0】

第 3 実施形態に係る多相型のメンバ関数定義の実体生成例を示す図である。

【図 1 1】

多相型の定義の従来例とその利用方法を示す図である。

【図 1 2】

実体化された多相型の定義の従来例を示す図である。

【図 1 3】

多相型の関数定義の従来例を示す図である。

【図 1 4】

従来のプログラム言語処理システムの構成を示すブロック図である。

【図 1 5】

従来のプログラム言語処理システムの処理の流れを示す概念図である。

【図 1 6】

従来の多相型の実現例を示す図である。

【図 1 7】

標準的なファイル構成における従来の多相型の定義方法例を示す図である。

【図 1 8】

多相型の関数定義をヘッダファイルに置いた従来例を示す図である。

【図 1 9】

多相型と同名のファイルに多相型の関数定義を置いた従来例を示す図である。

【図 2 0】

不必要な多相型のメンバ関数定義の実体が生成される従来例を示す図である。

【図 2 1】

複数のリロケータブルオブジェクトファイル中に多相型の関数定義の実体が重複して生成される例を示す図である。

【符号の説明】

- 1 ソフトウェア駆動装置
- 2 原始プログラム群

3 前処理装置

4 最適化処理装置

4 a 多相型定義情報テーブル

5 言語処理装置

6 リンク処理装置

7 目的プログラム

8 前処理後のファイル

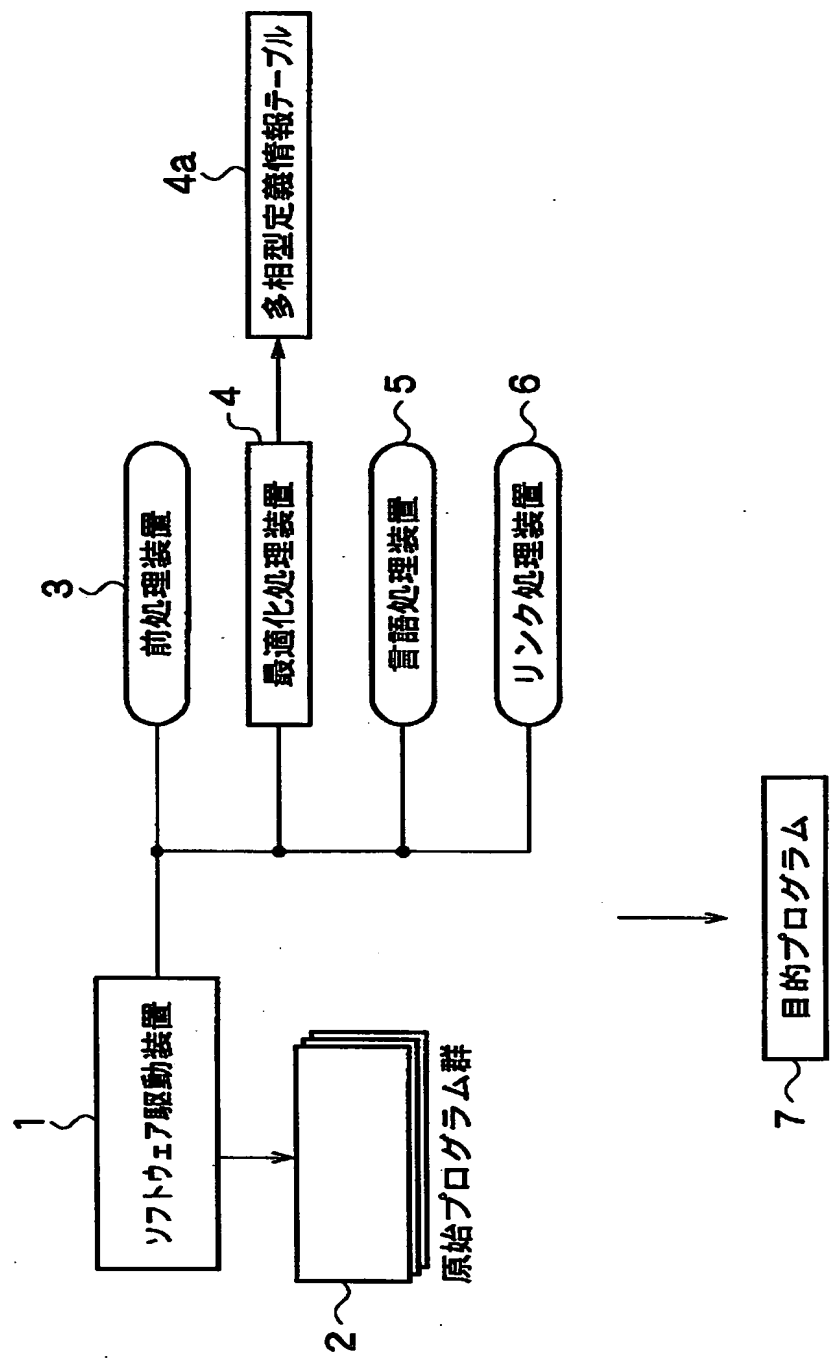
9 リロケータブルオブジェクト

f 0, f 1, f 2, f 3 前処理後の原始プログラム

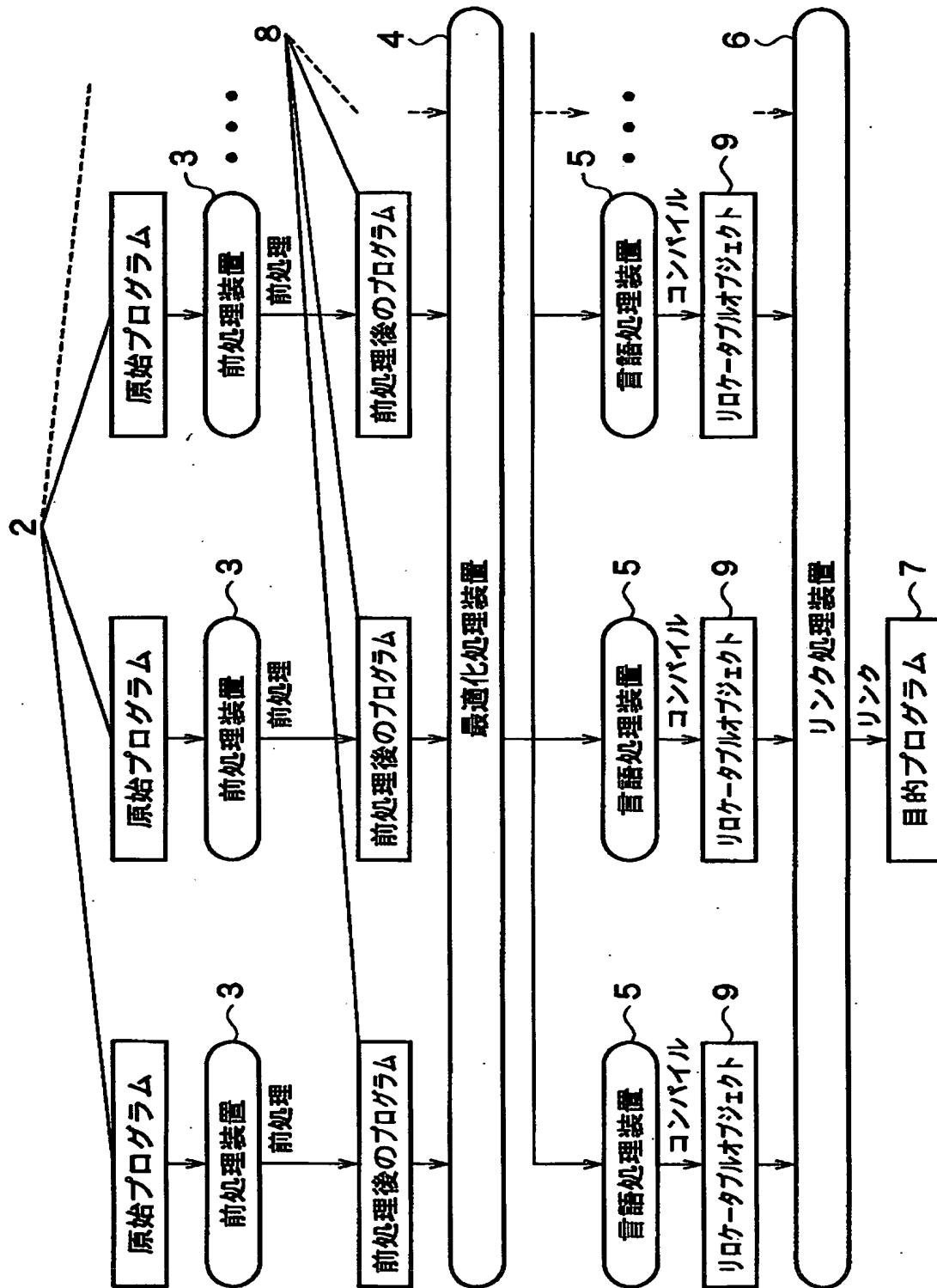
f 0' , f 1' , f 2' , f 3' 最適化後の原始プログラム

【書類名】 図面

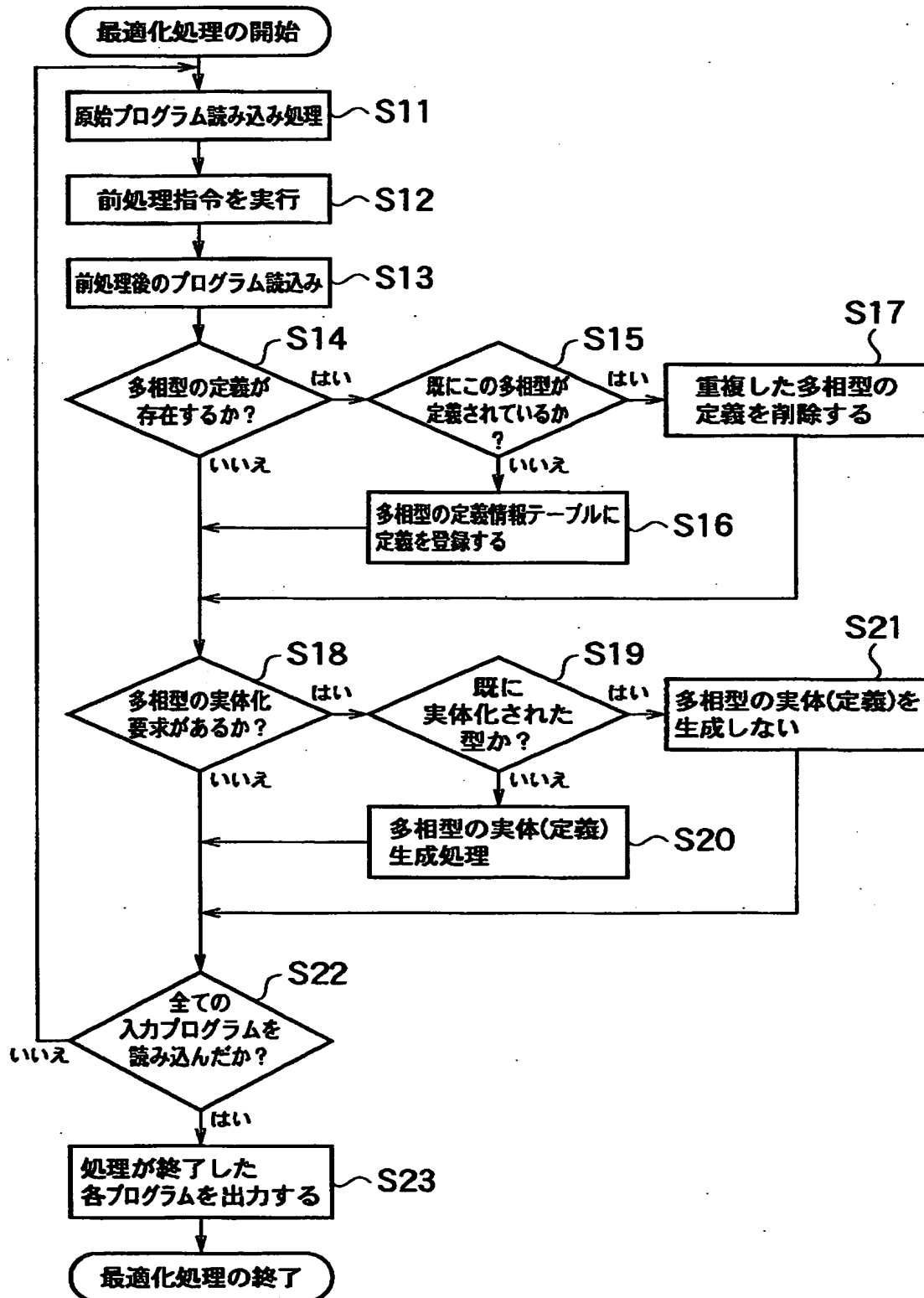
【図 1】



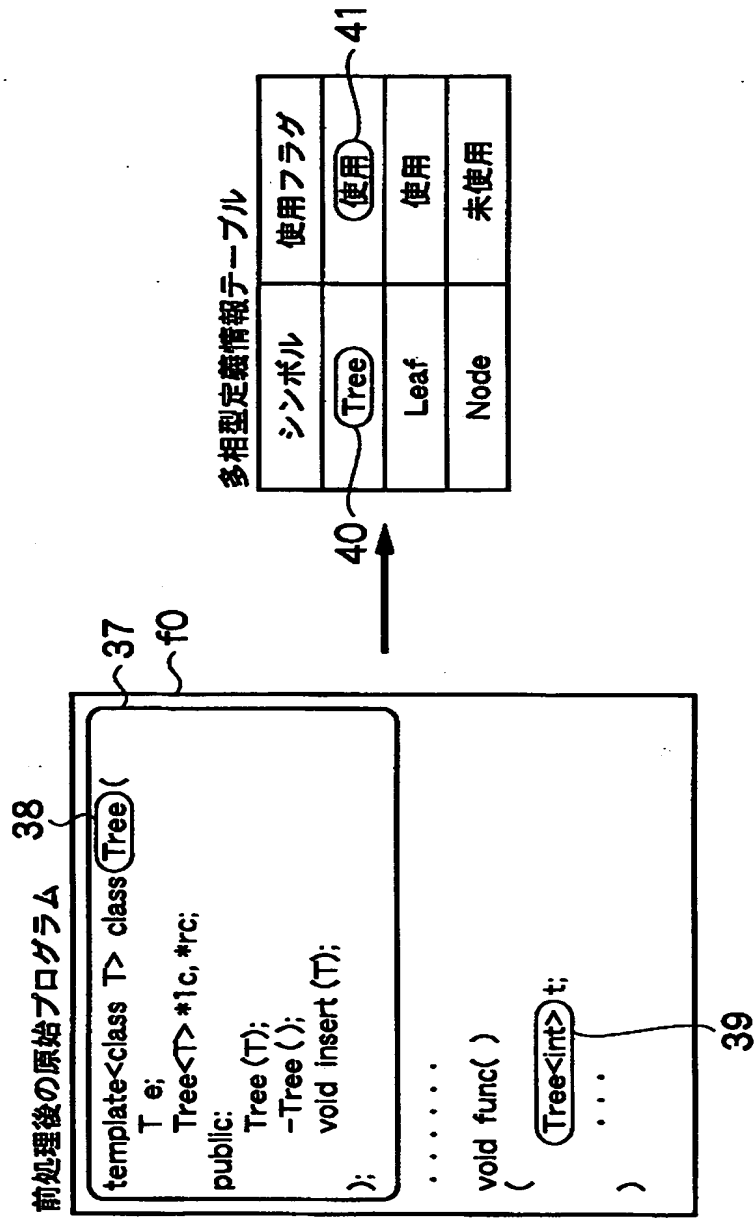
【図 2】



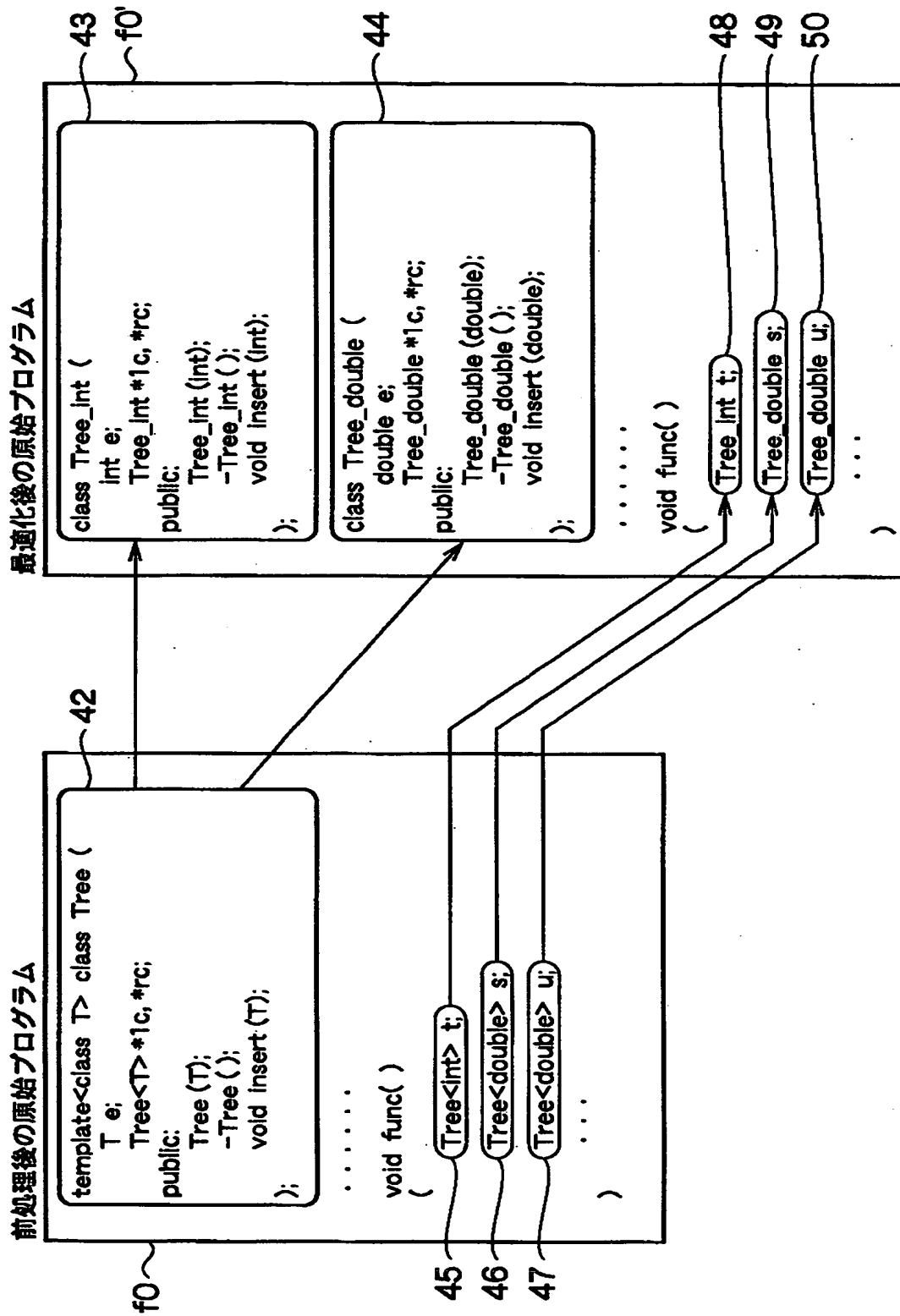
【図 3】



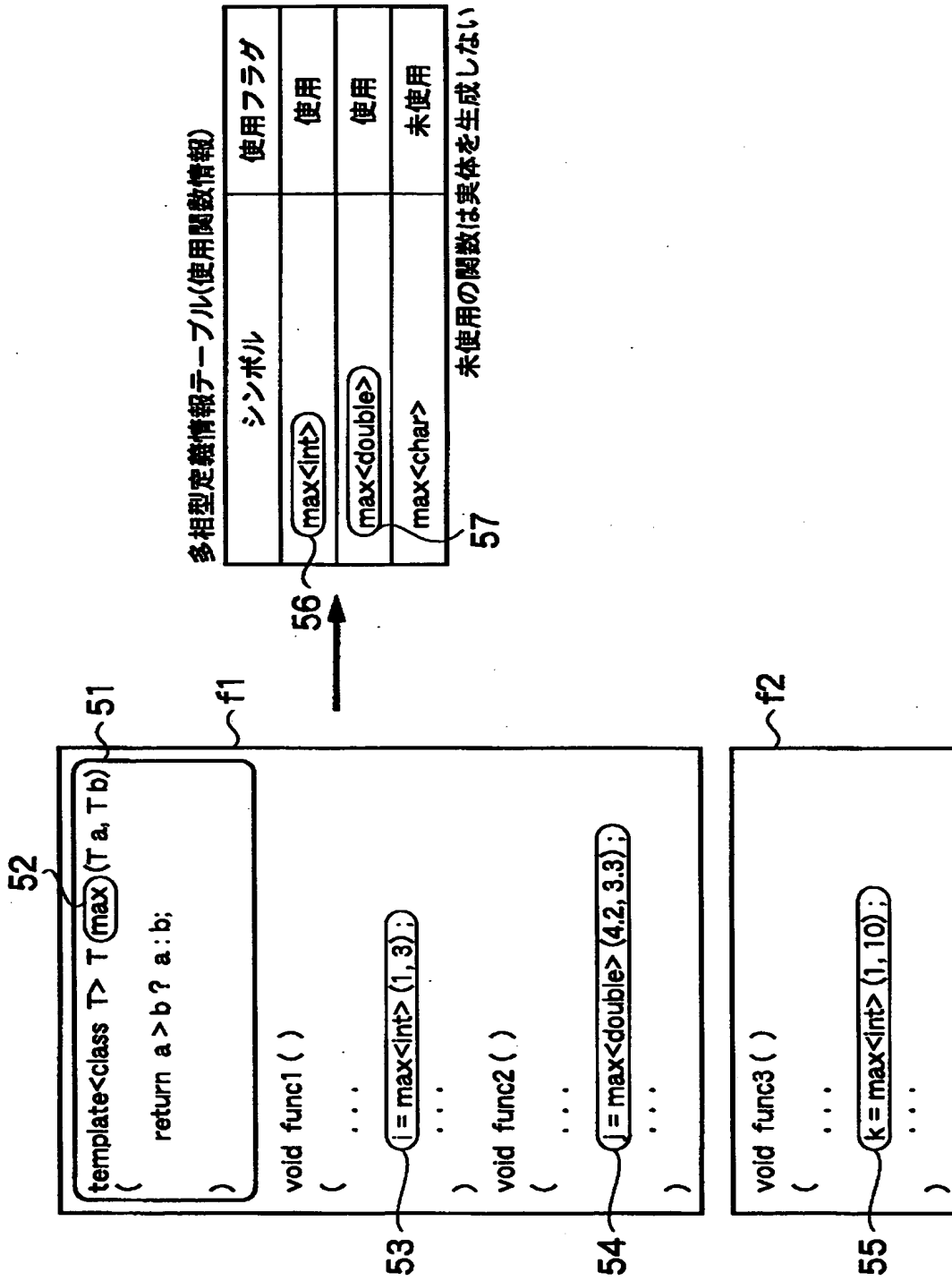
【図 4】



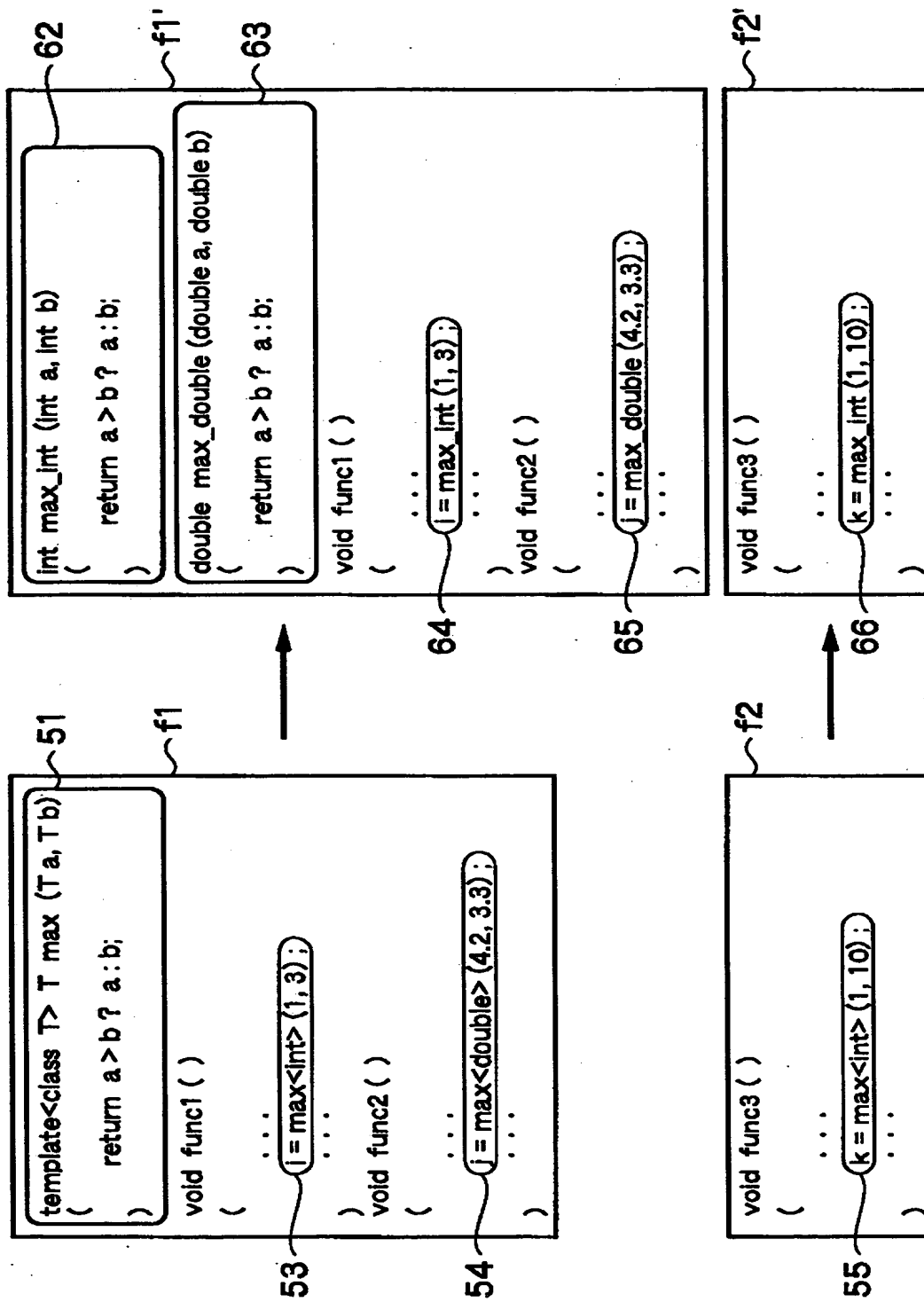
【図 5】



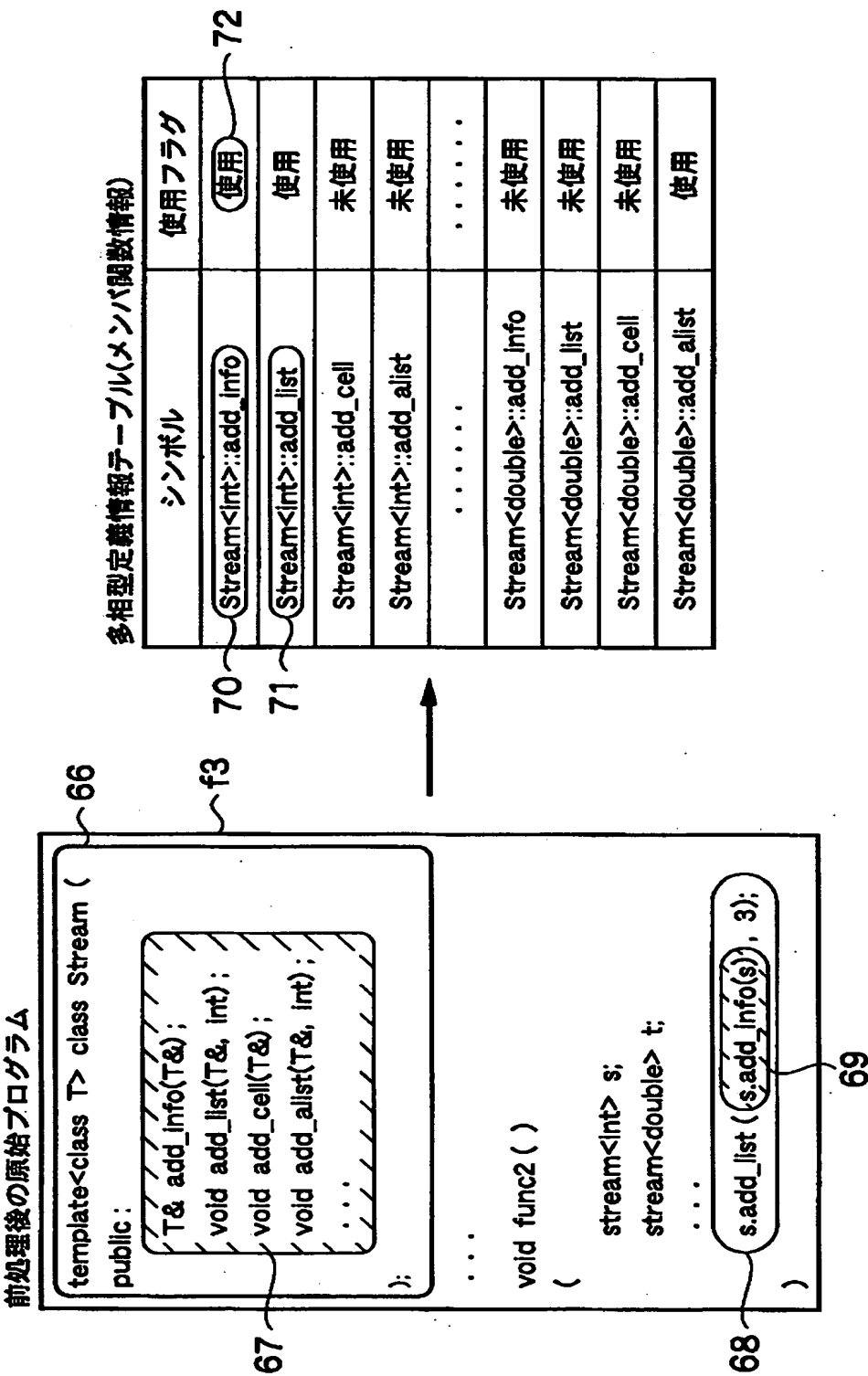
【図 6】



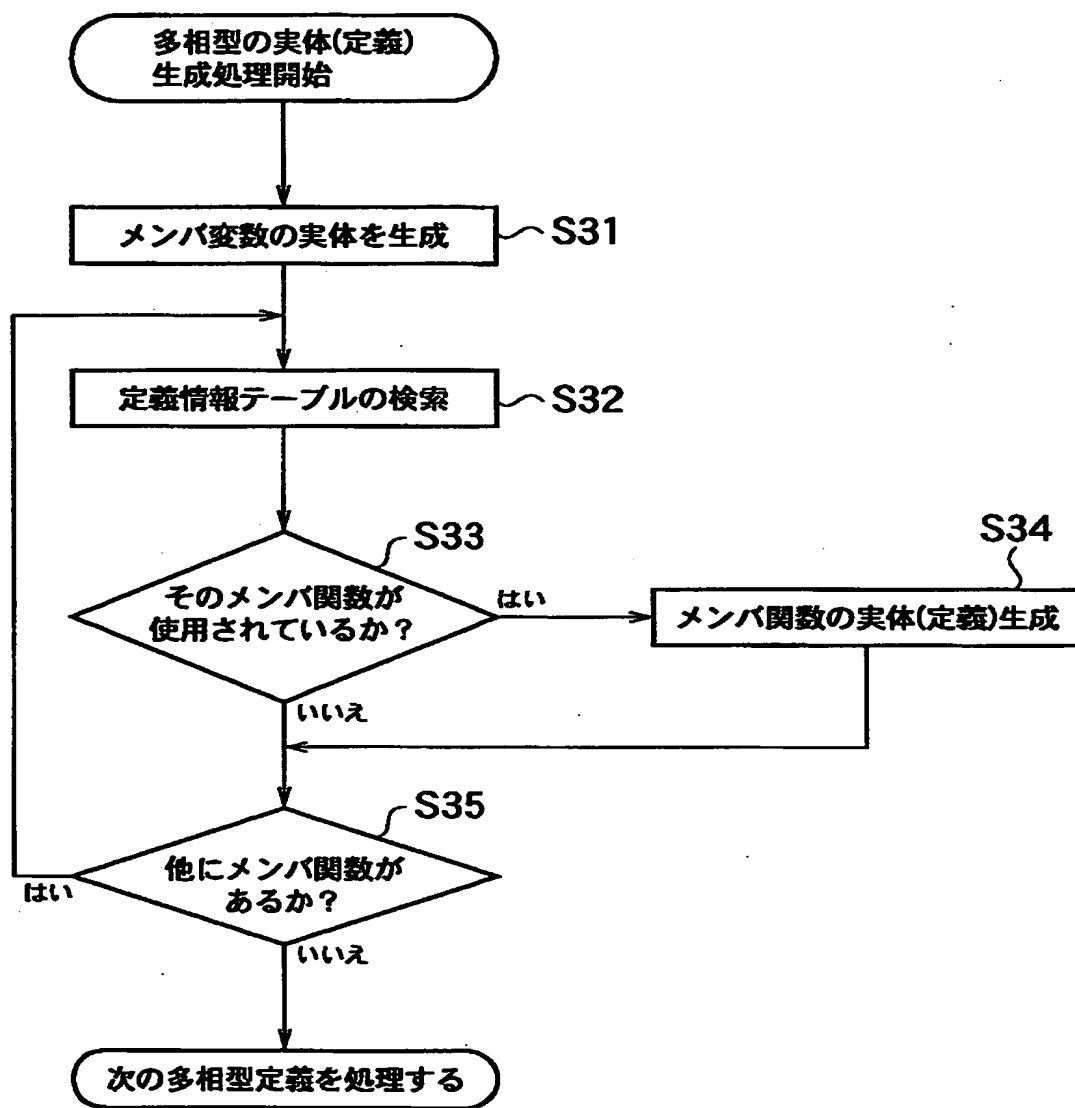
【図 7】



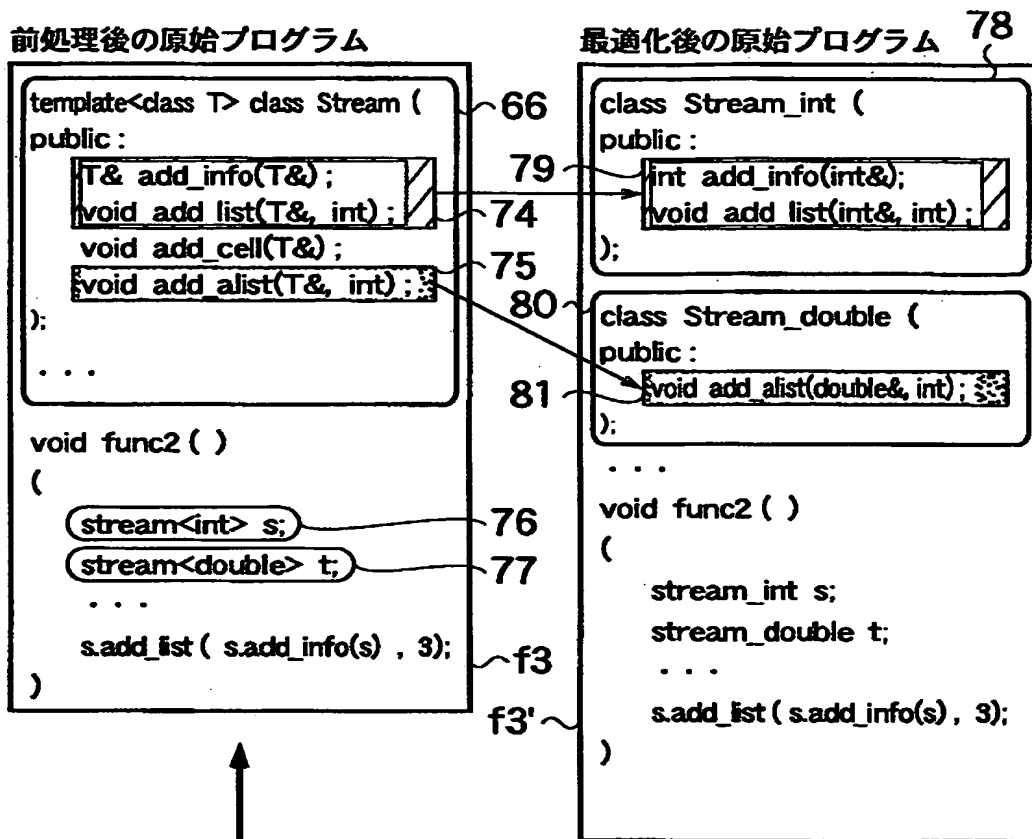
【図 8】



【図 9】



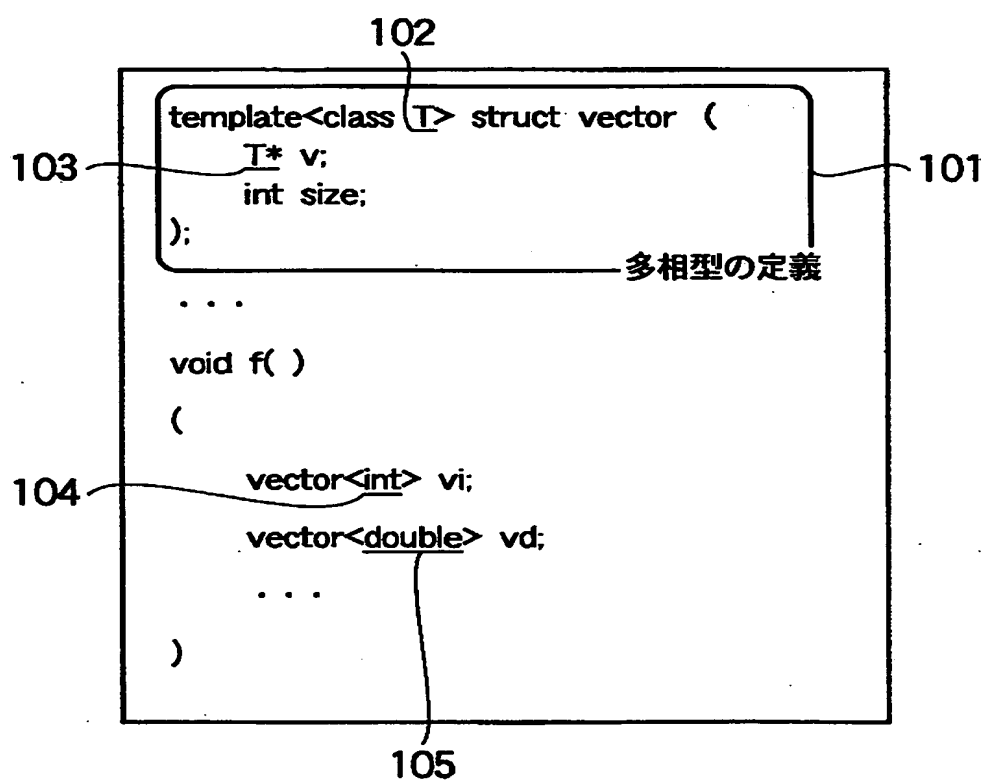
【図 1 0】



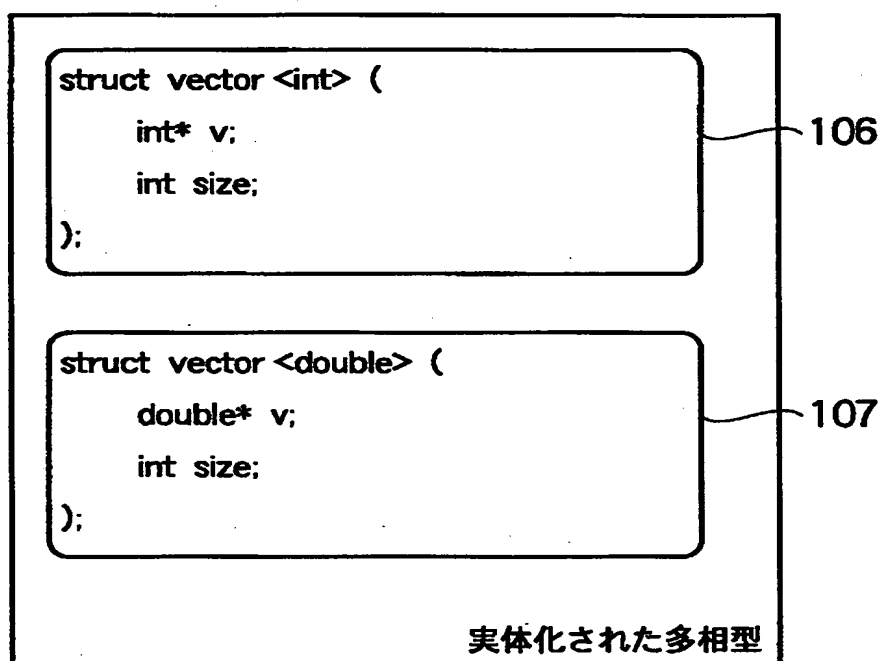
多相型定義情報テーブル(メンバ関数情報)

シンボル	使用フラグ
Stream<int>::add_info	使用
Stream<int>::add_list	使用
Stream<int>::add_cell	未使用
Stream<int>::add_alist	未使用
Stream<double>::add_info	未使用
Stream<double>::add_list	未使用
Stream<double>::add_cell	未使用
Stream<double>::add_alist	使用

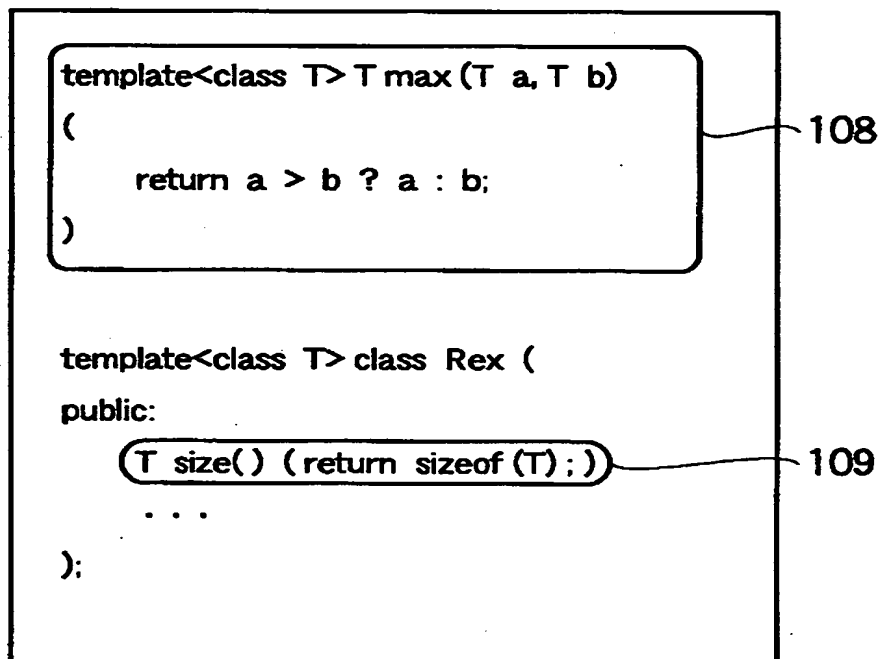
【図 1 1】



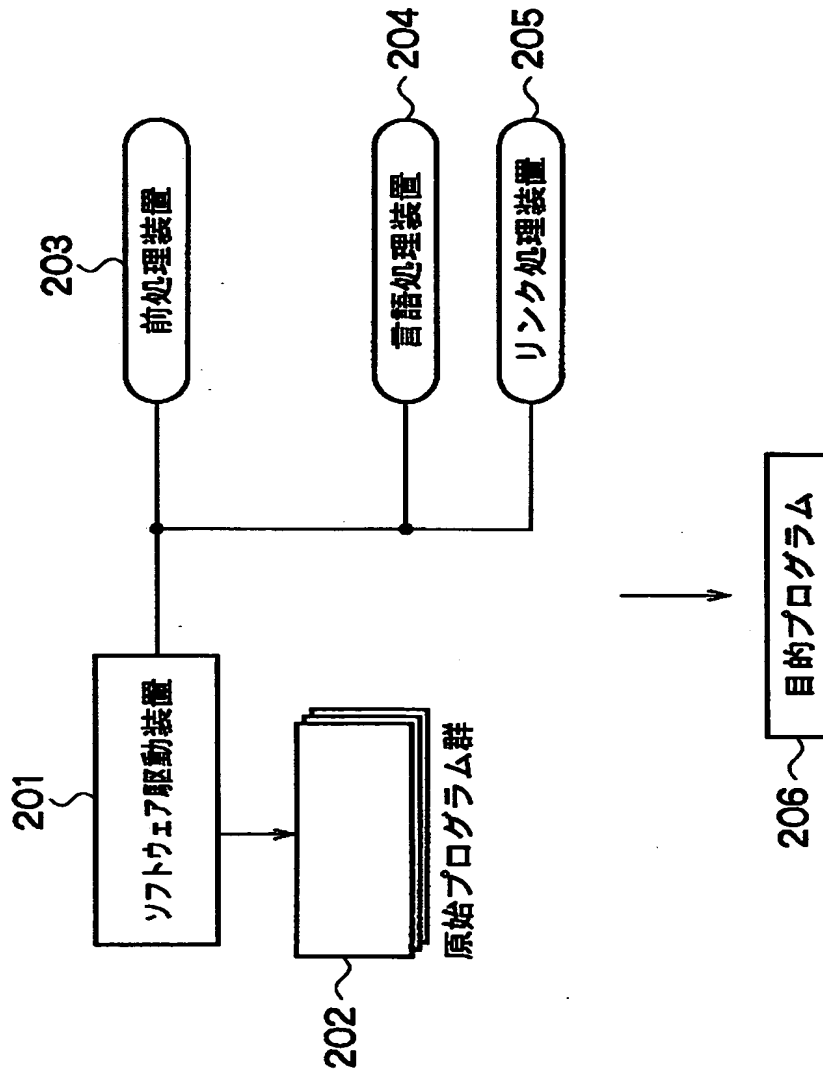
【図 1 2】



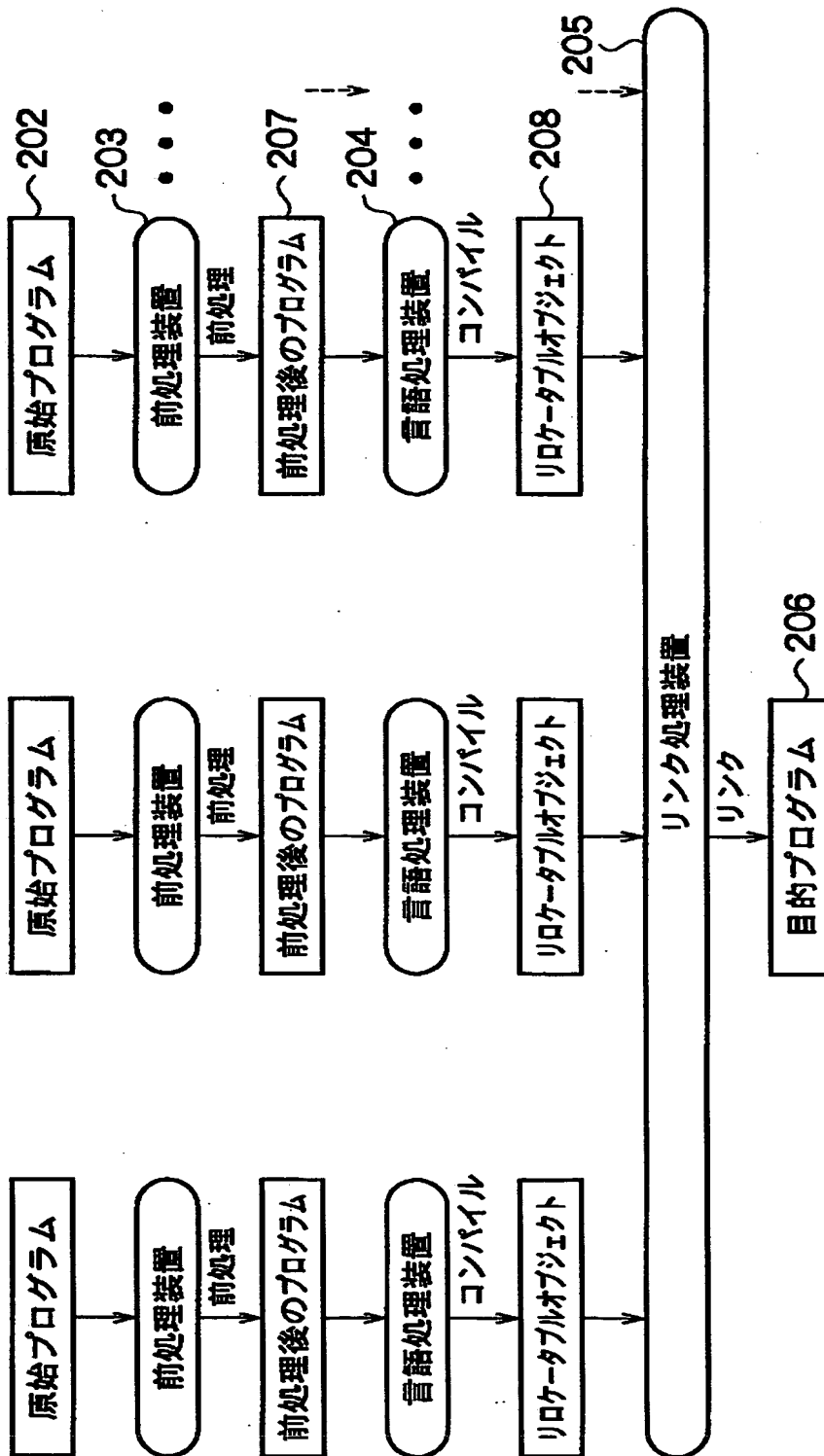
【図 1 3】



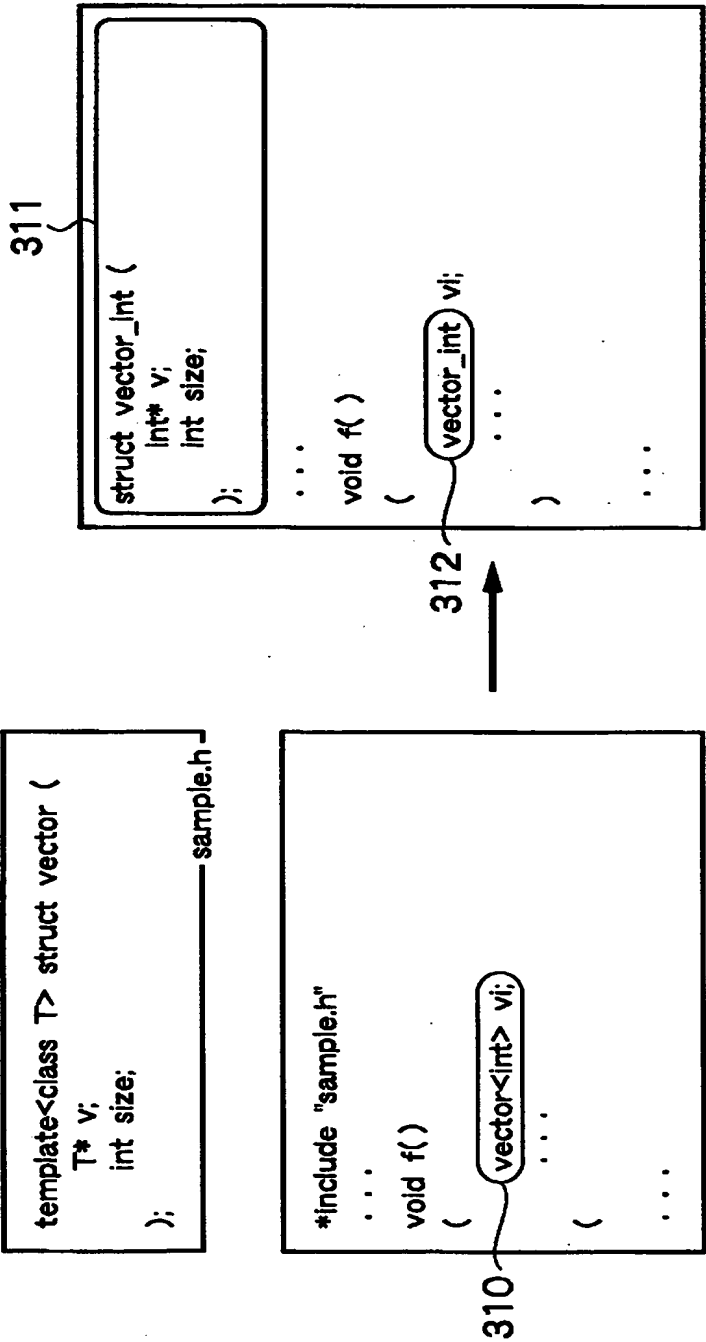
【図 1 4】



【図 1 5】



【図 1 6】



【図 1 7】

多相型の宣言ファイル(ヘッダファイル)

Stack.h ~

```
template<class T> class Stack (
    T* v;
    T* p;
    int size;
public:
    Stack(int);
    ~Stack();
    void push(T);
    T pop();
);
```

多相型の定義ファイル

313 ~

Stack.cpp ~

```
#include "stack.h"
template<class T> Stack<T>::Stack(int s)
(
    v = p = new T[size - s];
)
template<class T> Stack<T>::~Stack()
(
    delete[] v;
)
template<class T> void Stack<T>::push(T a)
(
    *p++ = a;
)
template<class T> T Stack<T>::pop()
(
    return *--p;
)
```

多相型を使用するソースファイル

314 ~

315 ~

file.cpp ~

```
#include "stack.h"
void func()
(
    Stack<int> s;
    ...
)
...
```

【図 1 8】

多相型の定義ファイル(ヘッダファイル)

Stack.h ~

```
template<class T> class Stack (
    T* v;
    T* p;
    int size;
public:
    Stack(int);
    ~Stack();
    void push(T);
    T pop();
);

template<class T> Stack<T>::Stack(int s)
(
    v = p = new T[size - s];
)
template<class T> Stack<T>::~ Stack()
(
    delete[] v;
)
template<class T> void Stack<T>::push(T a)
(
    *p++ = a;
)
template<class T> T Stack<T>::pop()
(
    return *--p;
)
```

316 ~

多相型を使用するソースファイル

```
317 ~ (*include "stack.h")
void func()
(
318 ~ Stack<int> s;
...
)
file.cpp ~
...
```

【図 1 9】

多相型の宣言ファイル(ヘッダファイル)

Stack.h

```
template<class T> class Stack (
    T* v;
    T* p;
    int size;
public:
    Stack(int);
    ~Stack();
    void push(T);
    T pop();
);
```

多相型の定義ファイル

319

Stack.cpp

```
#include "stack.h"
template<class T> Stack<T>::Stack(int s)
(
    v = p = new T[size - s];
)
template<class T> Stack<T>::~Stack()
(
    delete[] v;
)
template<class T> void Stack<T>::push(T a)
(
    *p++ = a;
)
template<class T> T Stack<T>::pop()
(
    return *--p;
)
```

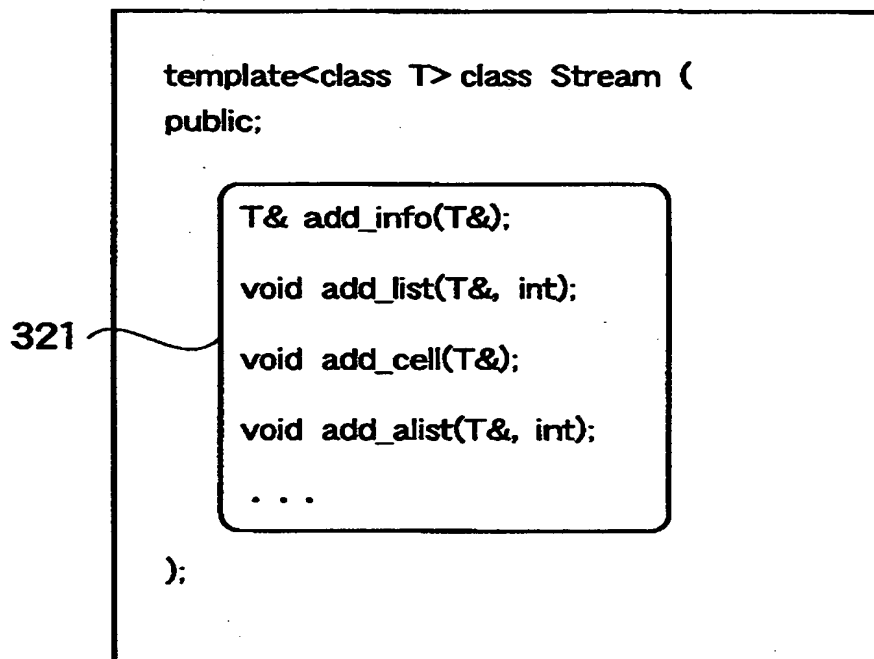
多相型を使用するソースファイル

318

file.cpp

```
#include "stack.h"
void func()
(
    Stack<int> s;
    ...
)
...
```

【図 20】



【図 2 1】

ah

```
template<class T> T max(T a, T b)
(
    return a > b ? a ; b;
)
```

f1.cpp

```
*include "a.h"
void func1 ( )
(
    ...
322 i = max<int>(1, 3);
    ...
)

void func2 ( )
(
    ...
323 j = max<int>(4, 3);
    ...
)
```

f2.cpp

```
*include "a.h"
void func3 ( )
(
    ...
324 k = max<int>(1, 10);
    ...
)
```

【書類名】 要約書

【要約】

【課題】 重複した不要な多相型の定義を取り除くことができると共に、不要な多相型の実体化を回避することができるプログラム言語処理システムのコード最適化方法を提供する。

【解決手段】 最適化処理装置 4 は、目的プログラム 7 を生成するために構成される全ての原始プログラム 2 に対し、多相型の情報を多相型定義情報テーブル 4 a に格納し、この多相型定義情報テーブル 4 a を参照して不要なコードを検知し、必要な定義のみ実体化し、各々の翻訳単位として言語処理装置 5 にプログラムを出力する。

【選択図】 図 1

出 願 人 履 歴 情 報

識別番号 [000003078]

1. 変更年月日	1990年 8月22日
[変更理由]	新規登録
住 所	神奈川県川崎市幸区堀川町72番地
氏 名	株式会社東芝